Project in

Networked Software Systems

(044169)

******DHT Firefox Extension

Project Book

**Supervisors:**

Mr. Ittay Eyal

**Team Members:**

Daniel Aranki

Hani Ayoub

October 2010 Technion Institute of Technology

Table of Contents

[Introduction 3](#_Toc281079709)

[What is a DHT? 3](#_Toc281079710)

[Acronyms & Terminology 4](#_Toc281079711)

[Project Progress 4](#_Toc281079712)

[Why Firefox? 5](#_Toc281079713)

[What statistics do we need to gather? 5](#_Toc281079714)

[High-Level Design 5](#_Toc281079715)

[High-Level components communication 5](#_Toc281079716)

[Server-Side Design 6](#_Toc281079717)

[Data-base 6](#_Toc281079718)

[Server Logic 7](#_Toc281079719)

[Plots and reports 8](#_Toc281079720)

[Personal Report 8](#_Toc281079721)

[How to see user’s personal report? 9](#_Toc281079722)

[Global Report 9](#_Toc281079723)

[Analysis 12](#_Toc281079724)

[Goal 12](#_Toc281079725)

[Try 1: Relying on the *Mean* duty time and the Standard Deviation issue 13](#_Toc281079726)

[Try 2: Static analysis 14](#_Toc281079727)

[Final Try: Dynamic analysis 15](#_Toc281079728)

[Conclusion 16](#_Toc281079729)

[Future projects suggestions 20](#_Toc281079730)

[References 21](#_Toc281079731)

# Introduction

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.

In order to give the answer for this question, this project book presents a road-map for collecting relevant information from Firefox users and analyzing them 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 coded using JavaScript, PHP, MySQL and Firefox extension platform (XUL and XBL) and is an open source project.

This project has two major parts. The first is implementing a Firefox extension, which its goal is gathering relevant information from the users, this part is described in the first chapters in this project book, which explains what information is gathered, high-level design of both server and client components of the extension and plots the extension provided after gathering the information. The second part is analyzing the information in order to try answering the question we’re dealing with: whether a DHT can be implemented using Firefox web browser or not. Last chapters in this project book, explains in details what analysis made to convince the reader how DHT can be implemented on Firefox. Analysis chapter includes a proposition of DHT layer with a dynamic algorithm that tries to efficiently use Firefox and its users to implement DHT.

## 

## 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:

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:

* ***User:*** defined as a person that installed the extension on one machine at least (user may install the extension over more than one machine).
* ***User ID:*** a unique identifier for each user.
* ***Node:*** a machine runs Mozilla Firefox browser with the extension installed.
* ***Node ID:*** a unique key for each node.
* ***Server:*** the machine that collects the data from *Nodes*.

## Project Progress

* First, an extension for Firefox that includes all the features stated in this document was published.
* Afterwards, the extension is spread to Firefox users over the internet.
* After enough statistics were gathered, both evaluation and analysis were held to decide whether a DHT can be run on Firefox, in sense of duty time, utilization and load, or not.

Answering the question of the ability to implement a DHT data-type under Mozilla Firefox in details.

## Why Firefox?

Mozilla Firefox is one of the most popular web browsers 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 has 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***: Operating system version and Firefox version. 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. Its responsibility is to manage the data gathered and communicate with user machines; each machine sends 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 information using Java-Script code. It sends the data it gathered/calculated to the server described above through the server interface; the server then collects and saves the data in its MySQL server (and may add some interpretations).

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

## Server-Side Design

As mentioned, the server has two parts, the Database, which is responsible for holding the statistics gathered, and the Logic that takes commands to save data and create dynamic reports.

## Data-base

The collected data are saved on server machine using MySQL server. Besides the collected information, the system keeps a unique ID for each user and a unique ID for each node (for each user) that installed the extension which will be determined by the server the first time the node connects the system (upon installation). Node ID identifies the machine every time it sends information to the server and is used for mapping purposes.

The server holds the following tables:

* ***Users table***: holds users information, has the following fields:
  + *User ID (MD5’ed [4] , all occurances of User IDs are MD5’ed)*
  + *Join Date*
* ***Nodes table:*** holds all nodes in the system, has the following fields:
  + *User ID*
  + *Node ID*
  + *Join date*
* ***Duty-Time table***: holds the times every machine opens/closes a FF session and the location information the user entered from. Field are as following
  + *Duty entry*
  + *Start*
  + *End*
  + *Node ID*
  + *Location Entry*
* ***Specifications table:*** for each node, machine’s specification info will be collected. Fields are:
  + *Node ID*
  + *Firefox version*
  + *OS*
* ***Location table:*** nodes’ location. Has the following fields:
  + *Location Entry*
  + *Node ID*
  + *IP*

The following chart, explains the info saved for a user:

*User statistics example*

## Server Logic

Beside the database, there lies a logic which is responsible for adding, removing, updating the database and creating dynamic reports and charts which are used for the analysis of the results. The logic is PHP coded, resides on the server side. The following is a brief description of its functionality:

* *Connect\Disconnect from the database*: used every time we need to change the DB.
* *Register for a new user*: used when a new user (means a user that downloaded the extension has registered as a new user to the system) joins the system. Adds his non-personal info to the DB such as mail (MD5’ed, new node ID and its info, etc…).
* *Gather statistics*: adds new information to the node the user is connected from such as start and end time.
* *Create personal summary report*: whenever the user asks for his personal statistics, this logic creates summary information for him (see example below). This report relies only on user’s personal info only.
* *Create global summary report*: for analysis usage, administrators use this logic to create a global to follow progress and analyze the statistics gathered (see example below). This report relies on ALL users’ info without providing any personal relationship to these pieces of information.

*Note: Why gathering and reporting logic resides on server (not client machines)?*

Gathering statistics and reporting logic resides on server machine in order to save consistent information. The problem in using client machine is the ***“Clock”***, which could operate under different time zones in different client machines. Using the server machine for gathering the information allows us to use the same Clock since there is only one server machine.

## Plots and reports

Server logic component dynamically generates reports and plots for each user (personal information report) to be viewed only from the extension for user usage, and global report that includes altogether statistics.

### Personal Report

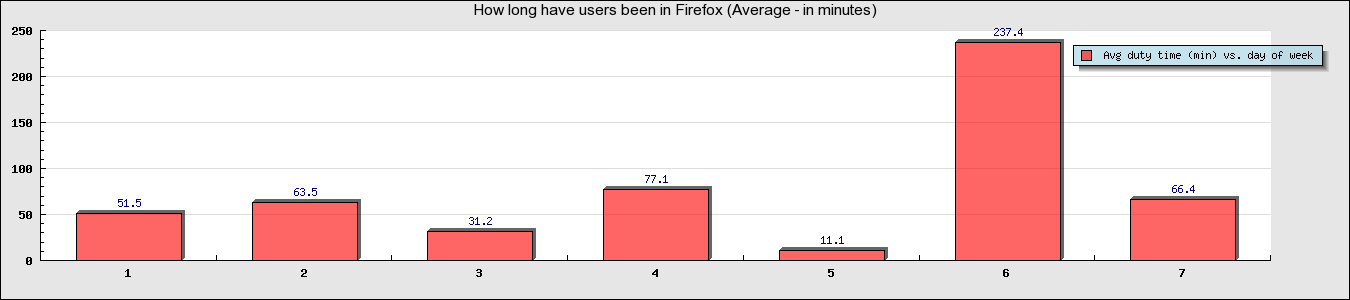
Each user has summary information that includes his average\min\max\total duty time, duty time standard deviation [7] and number of extension usages.



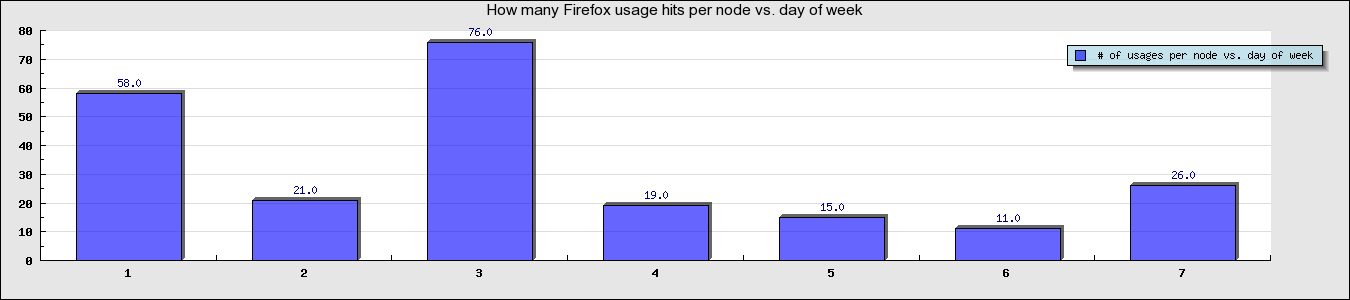
*Personal summary info – example*

Also, each user can view the following graphs:

1. Average duty time vs. day of week. This graph shows the average duty time for the user each day of week



1. Number of extension usage vs. WW. Similar to the graph above, but shows number of extension usages rather than average duty time



Average duty time and Number of extension usage vs. WW\Hour\Month (more 6 graphs) are also generated and can be seen from user personal report.

### How to see user’s personal report?

After a user installs the extension (link can be found in reference chapter), a new option called “Check your personal usage statistics!” will be added under “Tools” tab, clicking on it will open a new window with the desired personal report



*How to see user’s personal report*

### Global Report

Combining all statistics and info gathered from each user, a global report is generated dynamically. As for the personal report, the page has a summary info that includes total number of users, total number of nodes, Average\Min\Max duty time, duty time standard deviation and total number of extension usage

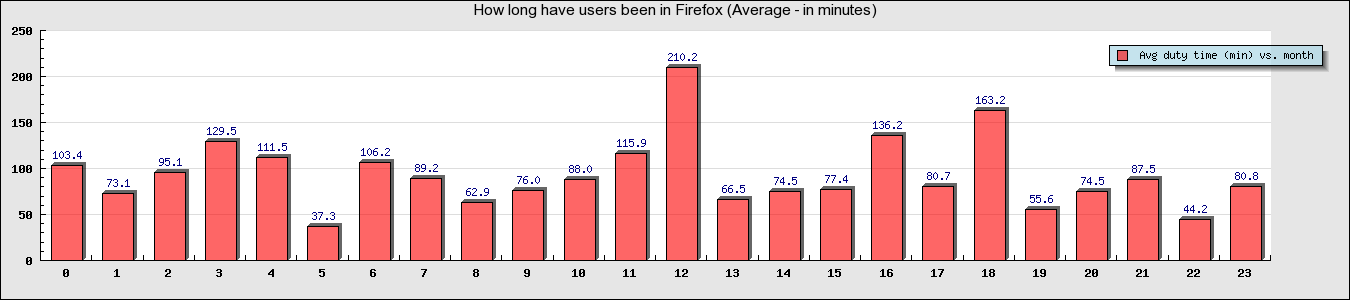


*Global summary info – example*

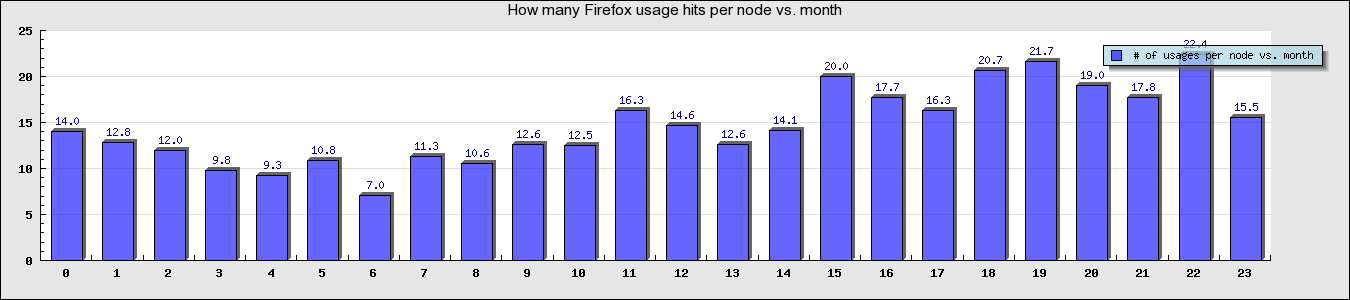
Also, there are two types of plots that can be generated in this page, fixed plots (as personal report plots) and variable plots that can be generated by the user after providing relevant variables

***Fixed plots:***

1. Total average duty time vs. hour. This graph shows the total average duty time of all users each hour of a day



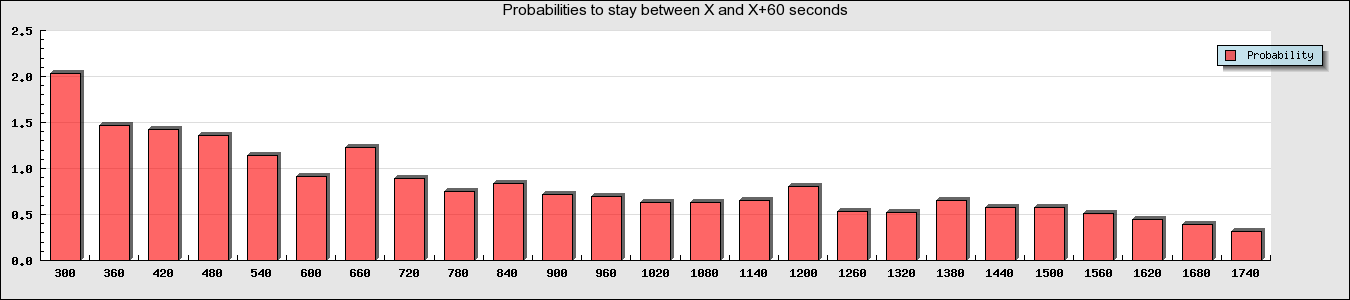
1. Total number of extension usage vs. hour. Similar to the graph above, but shows total number of extension usages rather than total average duty time



Total average duty time and Number of extension usage vs. WW\Day\Month (more 6 graphs) are also generated and can be seen from global report page [8].

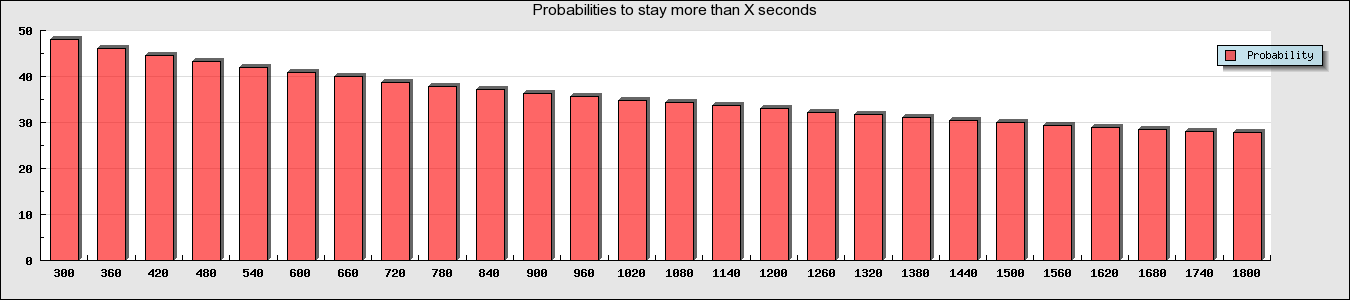
***Variable plots:***

1. Probabilities to stay in Firefox between given and (Duty time histogram). For example, for the following variables:

The following graph and table will be generated:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time | 300 | 360 | 420 | 480 | 540 | 600 | 660 | 720 | 780 | 840 | 900 | 960 | 1020 | 1080 | 1140 | 1200 | 1260 | 1320 | 1380 | 1440 | 1500 | 1560 | 1620 | 1680 | 1740 |
| Probability | 2.03 | 1.46 | 1.42 | 1.35 | 1.14 | 0.91 | 1.22 | 0.88 | 0.75 | 0.83 | 0.71 | 0.69 | 0.62 | 0.63 | 0.65 | 0.80 | 0.53 | 0.51 | 0.65 | 0.58 | 0.58 | 0.50 | 0.44 | 0.39 | 0.31 |

For instance, the probability that a user enters Firefox and stays between 5 min (300 sec) and 6 min (360 sec) is: ***2.03%***

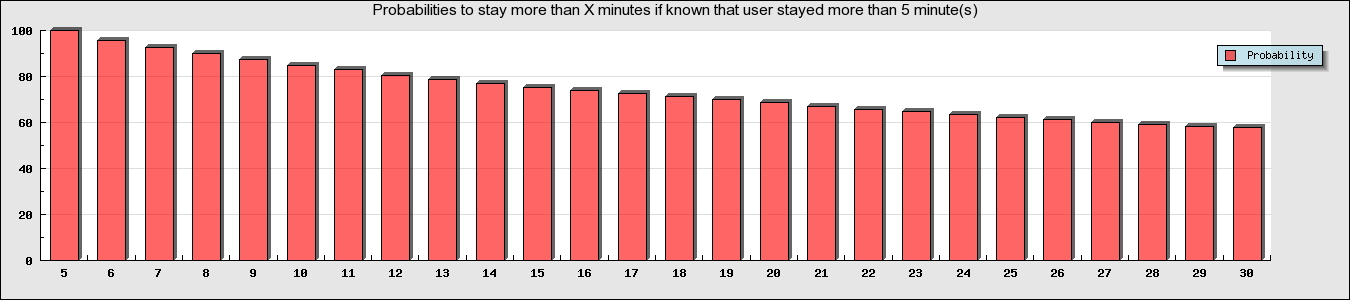
1. Accumulative probability to stay in Firefox for more than seconds. For example, with same previous example variables, the following graph and table will be generated:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time | 300 | 360 | 420 | 480 | 540 | 600 | 660 | 720 | 780 | 840 | 900 | 960 | 1020 | 1080 | 1140 | 1200 | 1260 | 1320 | 1380 | 1440 | 1500 | 1560 | 1620 | 1680 | 1740 | 1800 |
| Probability | 48 | 46 | 45 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 36 | 35 | 34 | 34 | 33 | 32 | 32 | 31 | 31 | 30 | 29 | 29 | 28 | 28 | 28 |

For instance, the probability that a user enters Firefox and stays for more than 15 minutes (900 seconds) is: ***36%***

1. Probabilities to stay more than minutes if it’s known that user stayed for more than minutes. For example, for the following variables:

The following graph and table will be generated:



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Minutes | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| Probability | 100 | 96 | 93 | 90 | 87 | 85 | 83 | 80 | 79 | 77 | 75 | 74 | 72 | 71 | 70 | 69 | 67 | 66 | 65 | 63 | 62 | 61 | 60 | 59 | 58 | 58 |

For example, the probability that a user stays for 10 min if he already stayed for 5 minutes is: ***85%***

# Analysis

## Goal

In order to answer the main question of the project, we need to analyze the statistics gathered. The optimal approach is trying to figure out the analytical distribution function that fits our statistics. X axis of this function is time and Y axis of the function is Probability

*Time*

*Probability*

*Example for Time-Probability desired function*

*Why are analytical functions good?*

If we derive an analytic function from our statistics, it would be easy to analyze the function rather analyzing the statistics gathered (in tables and numbers) especially if the function suits a known distribution functions such as Poisson[6] or Exponential[5] in terms of: finding the mean, maximum and minimum (not only for the function but for more complex conditional probabilities), draw sequential functions easily, drawing synoptic charts of 2 dimensional functions, easy predictions of probable events and many others.

In the following chapters we’ll describe the analysis process done try after try and phase after phase in order to reach the final analysis approach which provides the most valuable information to answer project’s question in the most possible detailed way.

## Try 1: Relying on the *Mean* duty time and the Standard Deviation issue

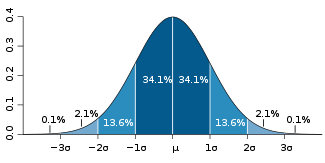
*What is standard deviation? [7]*

Standard deviation (ơ) is a widely used measurement of variability or diversity used in statistics and probability theory. It shows how much variation or dispersion there is from the average and is defined as:



Some widely used thumb rules (mainly for Normal distributions/Gaussian distributions):

* The probability for a measurement to be inside the ±ơ sleeve around the mean value is .
* The probability for a measurement to be inside the ±2ơ sleeve around the mean value is .
* The probability for a measurement to be inside the ±3ơ sleeve around the mean value is .



*A plot of a Normal distribution. Each colored band has a width of one* ơ*.*

*‘Small’* SD raises the confidence level of predicting the duty time of the next user (lowers the error rate). For instance: if SD=0 the theoretical prediction of the next user’s duty time is the exact mean duty time with a probability of 1. On the other hand, if SD is of the same order as the mean duty time then it would be hard to predict next user’s duty time (‘Large’ error rate). Relying on statistics gathered:

***Average duty time: 5382 seconds (~1.5 hours)***

***SD: 28474 seconds (~8 hours)***

As we can see, SD is greater than the mean duty time itself, which means that SD is ‘Large’ and thus the prediction level is low. Therefore, Try 1: relying on the mean time is unconfident because of the large SD.

Also, finding an analytical pre-analyzed probability distribution was not possible because the current (static) distribution lies between the exponential distribution and the Gaussian distribution (it is more like an exponential distribution but with ‘calmer’ drop than an exponential distribution).

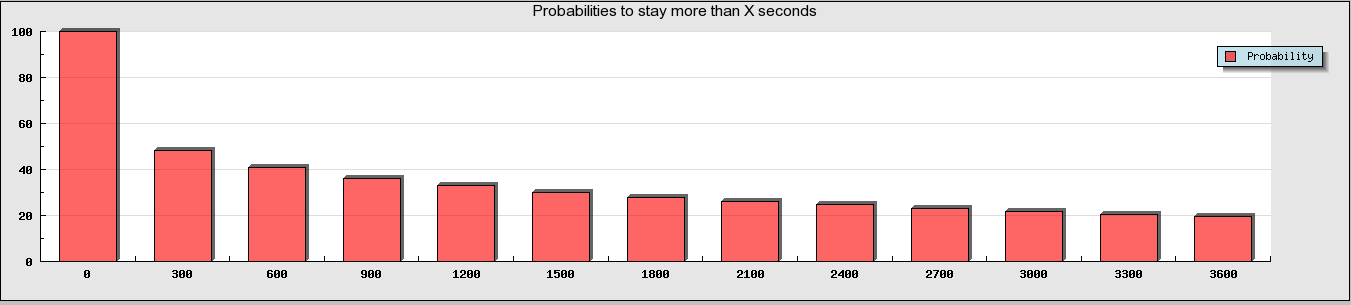
Moreover, analyzing statically is O.K but weak, since we can use the dynamic information to determine the behavior of the DHT itself as will later be explained in the final try.

## Try 2: Static analysis

The histogram provides good indication and intuition on the duty times limits. That is, what percent of the nodes used Firefox between X seconds and X+offset seconds. As mentioned, this is a good indication for the probability distribution that the Firefox users present. But a more interesting piece of information is: what percentage of users uses Firefox for **more** than X seconds. This is referred to, in the theory of statistics, as the accumulative probability function (or more accurately, the inverse accumulative function) which denotes:

This piece of information gives us a prediction of how many users will use Firefox for more than a specific period of time, which will further allow us to determine what uses can a DHT, if implemented under Firefox, be good for.

Example:

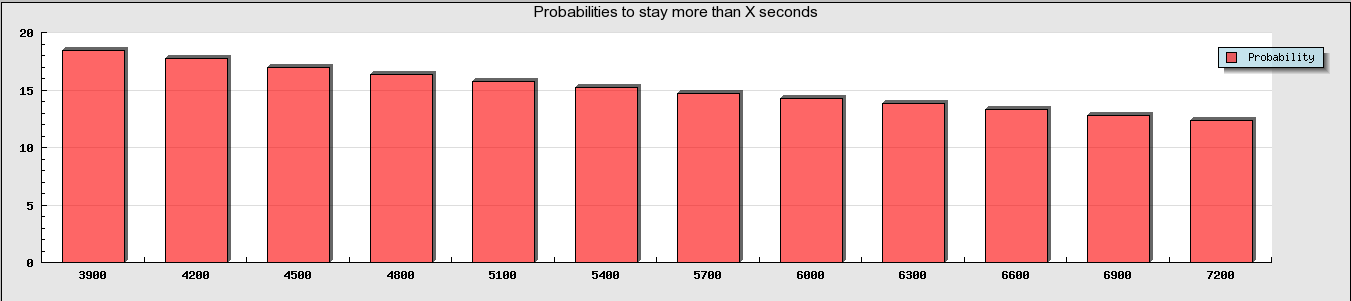


*The plot of the inverse accumulative distribution for the statistics we gathered, between 0 and 1 hour with an offset of 5 minutes.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X | 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 |
| Probability to stay more than X minutes | 100 | 48 | 41 | 36 | 33 | 30 | 28 | 26 | 25 | 23 | 22 | 21 | 20 |

*The table with the values of the inverse accumulative probabilities for each X.*

As spotted from the probabilities above, 1 user out of every 5 users uses Firefox for at least 1 hour. If we consider the graph from 65 minutes up to 120 minutes with 5 minutes offset we get:



*The plot of the inverse accumulative distribution for the statistics we gathered, between 0 and 1 hour with an offset of 5 minutes.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X | 65 | 70 | 75 | 80 | 85 | 90 | 95 | 100 | 105 | 110 | 115 | 120 |
| Probability (out of 100%) to stay more than X minutes | 18 | 18 | 17 | 16 | 16 | 15 | 15 | 14 | 14 | 13 | 13 | 12 |

*The table with the values of the inverse accumulative probabilities for each X.*

The probabilities are relatively high and are good for somewhat heavy DHT usages. But the following question arises, how can we raise our confidence level in knowing which user will stay further more in Firefox (and thus give him a heavier DHT data) and which user will stay little more in Firefox (and thus start to duplicate the data he has to other nodes, or simply give him lighter data). In order to answer this question, we went to analyze in a third try (dynamically, not only statically).

## Final Try: Dynamic analysis

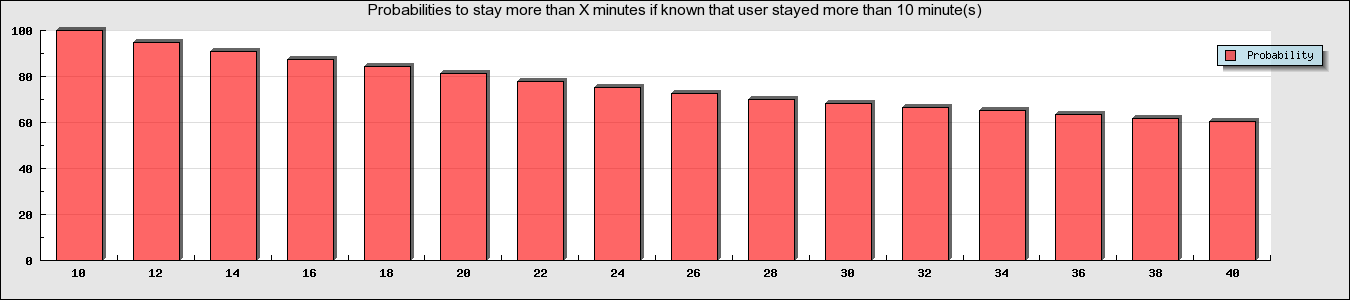
*What do we really need from the statistics?*

DHT is a dynamic protocol for distributing information across nodes. Predicting duty time for node/user that enters the system is the most important criterion for deciding what to do with it. Therefore, what we really need from the statistics is: *predicting duty time for a user that entered the system*. However, inverse accumulative probability of users with greater duty time than T helps predicting the total time a user will stay in FF; this information is static and doesn’t use statistics in optimal way. In order to add the dynamic behavior to the system the kind of questions we need to answer is: *given that a user has been in FF for time, what is the probability for the user to stay more than time?* Answering such questions is informative since it will help DHT implementer to decide:

* Node degree
* When a node becomes ready to join DHT graph.
* What kind of DHT (heavy/light data sharing, etc..) the node is suitable for
* Minimizing data loss

*How can we derive this info from the statistics?*

Using conditional probability definition, what we need to find is:

Example:

*The plot of the conditional probability (explained above) for a user to stay more than Xend minutes given that a node stayed in FF for more than 10 minutes (for Xend between 10 minutes and 40 minutes with an offset of 2min)*

*The table with the values of the conditional probabilities for each X.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (minutes) | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 | 38 | 40 |
|  | 100 | 95 | 91 | 87 | 84 | 81 | 78 | 75 | 72 | 70 | 68 | 66 | 65 | 64 | 62 | 60 |

For instance, if a node stayed for more than 10 minutes in FF, the probability to stay for 20 minutes (10 minutes further) is 81%. This, for example, gives a high confidence level that the user will stay 10 more minutes (than predicting the user will stay more than 10 minutes from the beginning. Which is, according to try 2, 41%) and enables the DHT to give this node heavier data with less probability of data loss.

## Conclusion

After the three analysis phases that we held and according to the gathered statistics, we can confidently say that a DHT data structure can be implemented in FF with a variety of uses that can be partitioned to the nodes dynamically (in run-time).

In order to extract the maximum out of the gathered data, a proposed sketch of the DHT behavior is described below:

The structure of the DHT we are talking about contains several overlay networks (refer to the section [What is a DHT?](#_What_is_a) For the definition) each with different level of data weights (data weight is an indication on how large the files are in size). Such that, the more confident we are that a node will stay ‘long enough’ (relative to the data weight) we will attach him to an overlay network that holds such heavy data and vice versa.

The scenario starts when a user enters FF. DHT should not attach the user’s node to any DHT graph, why? In the 2nd try of the analysis, we’ve shown that more than 50% of the nodes leave FF before staying 5 minutes, thus, it would be irrelevant to attach them to any DHT graph (more specifically, in case the data transfer time will take more than 5 minutes). The arising question is therefore, how long should DHT wait before attaching a node to an overlay network? The tradeoff is as follows: if DHT waits too short, it can lose too many nodes too early (similar to the example above). On the other hand, if DHT waits too long, it can lose potential nodes for light uses. Such should be determined by (1) the lightest DHT use, that will provide an indication of the maximal accepted, and (2) a maximal acceptable probability of that will provide an indication of the minimal accepted (which yields minimal accepted). The value of should be selected between these two bounds.

Another piece of criterion to be set is the minimal accepted probability. This criterion enables DHT to determine when to attach the node to a heavier overlay network in a way that it considers a nominal time a node should stay online in order to be attached to that specific overlay network. If after staying that nominal time the probability to stay T more seconds (where T is the minimal time needed to transfer a file with maximal file-size from that overlay network) is not achievable under the minimal accepted probability that we chose then the DHT should choose not to attach the node to the overlay network, and if the probability (to stay online for enough time after that nominal time) is higher than this minimal probability, the DHT will choose to attach the node to the overlay network.

This minimal accepted probability () is controlled by the following tradeoff: the smaller the probability , the farther the “control checkpoints” of the DHT on the node will be and thus the more nodes being lost between those checkpoints; and the larger the probability , the larger the number of DHT partitions to overlay networks (but also the more confident we are about data losses).

and combined give an upper limit on the weight of lightest overlay network. From assuming a minimal accepted probability of the user will “most probably” stay online for another time (according to the statistics). This, considering an average transfer speed of the network should be enough time to transfer the largest piece of data in the lightest overlay network (or otherwise there’s no sense in attaching **any** node in the first place).

At this point, the following DHT parameters (of our proposed implementation) were decided:

* The heaviest data level overlay network (personal need)
* The lightest data level overlay network
* which indicates the number of overlay networks (partitions)

|  |
| --- |
| *Heaviest Overlay Network* |
|  |
|  |
| *…* |
| *…* |
| *Lightest Overlay Network* |

*Time Line*

*Illustration explains the overlay networks partitioning and how in Time-Line a user can transfer between them according to his current duty-time*

According to the the DHT will know when to “checkpoint” the node such that the checkpoint number *i* will be held time after the last checkpoint. Each satisfies the following: (Sigma on all k<i). If the next checkpoint is “large enough” for the next data weight level overlay network then the DHT will decide to attach the node to it, otherwise, it won’t.

***Example:***

The following example may is illustrative and may not be of a real usage. Also, please note that though the whole analysis that we’re doing here is static, but the DHT can change its decisions dynamically according to the most updated statistics data it has.

Let us assume we need a DHT that will support *sensitive* data of the sizes *30MB-100MB*. Let us also assume that the average transfer speed in the DHT network is *0.1MB/sec*. This means that the nominal transfer time for a 30MB file is *300 seconds* (i.e. 5 minutes). Because the data is sensitive, we can’t afford a probability of losing a node without preparation that is larger than *20%*. In other words, we need to be *80%* certain (at least) that the node will be able to transfer its data completely before losing the connection. This also yields

That’s why we can’t let the user join the lightest overlay network directly because there is a probability of more than *50%* that the node will drop before completing the transfer of the smallest file (30MB). In terms of probabilities: . So we will pick the earliest that satisfies:

According to our statistics, this yields: because: and we have good confidence (according to the level of *sensitivity* we defined above) that no more than 20% of the nodes will be lost before completing the complete data transfer (statistically).



|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X (minutes) | 2.5 | 3 | 3.5 | 4 | 4.5 | 5 | 5.5 | 6 | 6.5 | 7 | 7.5 |
| Probability | 100 | 97 | 94 | 92 | 90 | 88 | 86 | 84 | 83 | 82 | 80 |

Top of Form

Bottom of Form

|  |
| --- |
|  |

*The probability graph and table taken from global dynamic created graphs that shows that*

So now we know that the node will be attached to the lightest overlay network (of files with the maximum size of 30MB) after 2.5 minutes of himself opening the FF session. The next checkpoint is going to be after 5 minutes from that event. That is 7.5 minutes since opening his FF session (). According to our statistics which satisfies: is . In 9 minutes, we can transfer a file up to 54MB (according to the average transfer speed) so we can set the second overlay network’s file size limit to 54MB. And we can rest assured that our *sensitivity* requirement is met. So the next checkpoint will be 16.5 (7.5+9) minutes after opening the FF session which means 9 minutes after the previous checkpoint.



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | X (minutes) | 7.5 | 8.5 | 9.5 | 10.5 | 11.5 | 12.5 | 13.5 | 14.5 | 15.5 | 16.5 | | Probability | 100 | 97 | 94 | 92 | 89 | 87 | 85 | 83 | 82 | 80 | |

*The probability graph and table taken from global dynamic created graphs that shows that*

According to the same methodology, yields . In 12.5 minutes we can transfer a file up to 75MB (according to the average transfer speed). So the third overlay network file size limit is 75MB. We continue and we get and thus the fourth overlay network’s file size limit is: 102MB and we can transfer the largest file which is 100MB.

To conclude, in our example according to the current statistics data:

|  |  |  |
| --- | --- | --- |
| *Parameter* | *Meaning* | *Value* |
|  | *The time that needs to pass before the node gets attached to the lightest DHT overlay network* | 2.5 minutes |
|  | *The time between joining the lightest DHT overlay network and the first checkpoint* | 5 minutes |
|  | *The time between the first and the second checkpoints* | 9 minutes |
|  | *The time between the second and the third checkpoints* | 12.5 minutes |
|  | *The time between the third and the fourth (last) checkpoints* | 17 minutes |
|  | *The file size limit of the first overlay network (lightest)* | 30MB |
|  | *The file size limit of the second overlay network* | 54MB |
|  | *The file size limit of the third overlay network* | 75MB |
|  | *The file size limit of the fourth overlay network (heaviest)* | 100MB ***(target)*** |

# Future projects suggestions

Future projects that can be held depending on this project are briefly described below:

1. ***Implement the extension over other web-browsers:***

As said before, Mozilla Firefox has recorded to be responsible for 22.48% of the total usage share of web browsers. In order to cover a wider audience, the extension can be implemented over other web-browser such as Internet Explorer and Chrome. Server-side database and logic can be (and even should be) the same one, what needs to be implemented is the extension logic (which even can include shared logic from different extensions) on different browsers. Implementers should be aware of browser specific statistics that maybe should be gathered.

1. ***Implement DHT:***

Since DHT - as explained in the analysis chapter - can be implemented in Firefox, a big challenge is implementing it! We’ve tried to describe an implementable DHT with dynamic algorithm to decide how to manage the users for different weight (light and heavy) overlay networks; it can be a good reference. Moreover, a different challenge is thinking about implementing a DHT with self-learning component that depends on the statistics gathered dynamically. Such DHT - we believe - could be very useful and efficient.

# 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.

[4] Wikipedia. MD5

<http://en.wikipedia.org/wiki/MD5>

[5] Wikipedia. Exponential distribution

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

[6] Wikipedia. Poisson distribution

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

[7] Wikipedia. Standard deviation

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

[8] Link to the extension statistics

<http://ffstatistics.juniorhosting.net/>