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

[Abstract 3](#_Toc279347649)

[General description 3](#_Toc279347650)

[What is a DHT? 3](#_Toc279347651)

[Acronyms & Terminology 4](#_Toc279347652)

[Scenario description 6](#_Toc279347653)

[Project Progress 6](#_Toc279347654)

[Why Firefox? 6](#_Toc279347655)

[What statistics do we need to gather? 6](#_Toc279347656)

[High-Level Design 8](#_Toc279347657)

[High-Level components communication 8](#_Toc279347658)

[Server-Side Design 9](#_Toc279347659)

[Data-base 9](#_Toc279347660)

[Server Logic 10](#_Toc279347661)

[Personal Info report (example) 11](#_Toc279347662)

[Global Report (Example) 12](#_Toc279347663)

[Analysis 14](#_Toc279347664)

[Known probability distributions 14](#_Toc279347665)

[Poisson distribution [6] 14](#_Toc279347666)

[Exponential Distribution [5] 15](#_Toc279347667)

[Goal 16](#_Toc279347668)

[Try 1: Relying on the *Mean* duty time and the Standard Deviation issue 17](#_Toc279347669)

[Try 2: Choose known probability distribution function to fit for the statistics 18](#_Toc279347670)

[Final Try: 19](#_Toc279347671)

[References 20](#_Toc279347672)

# 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 coded using 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:

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 collects the data from *Nodes*.

# 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***: 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 residing in the Technion’s Networked Software Systems lab. 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:

## 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 and node joins the DHT 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 IDs MD5ed [4], has the following fields:
  + *User ID*
  + *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 (IP) the user entered from. Field are as following
  + *Duty entry*
  + *Start*
  + *End*
  + *Node ID*
  + *Location (IP)*
* ***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’s a logic sits behind it which is responsible for adding, removing, updating the database and creating dynamic reports and charts which is used for the analysis of the results. The logic PHP coded, resides on server side, following is 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 and added a new mail to the DB) joins the system. Adds his non-personal info to the DB such as mail (MD5ed, 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 info.

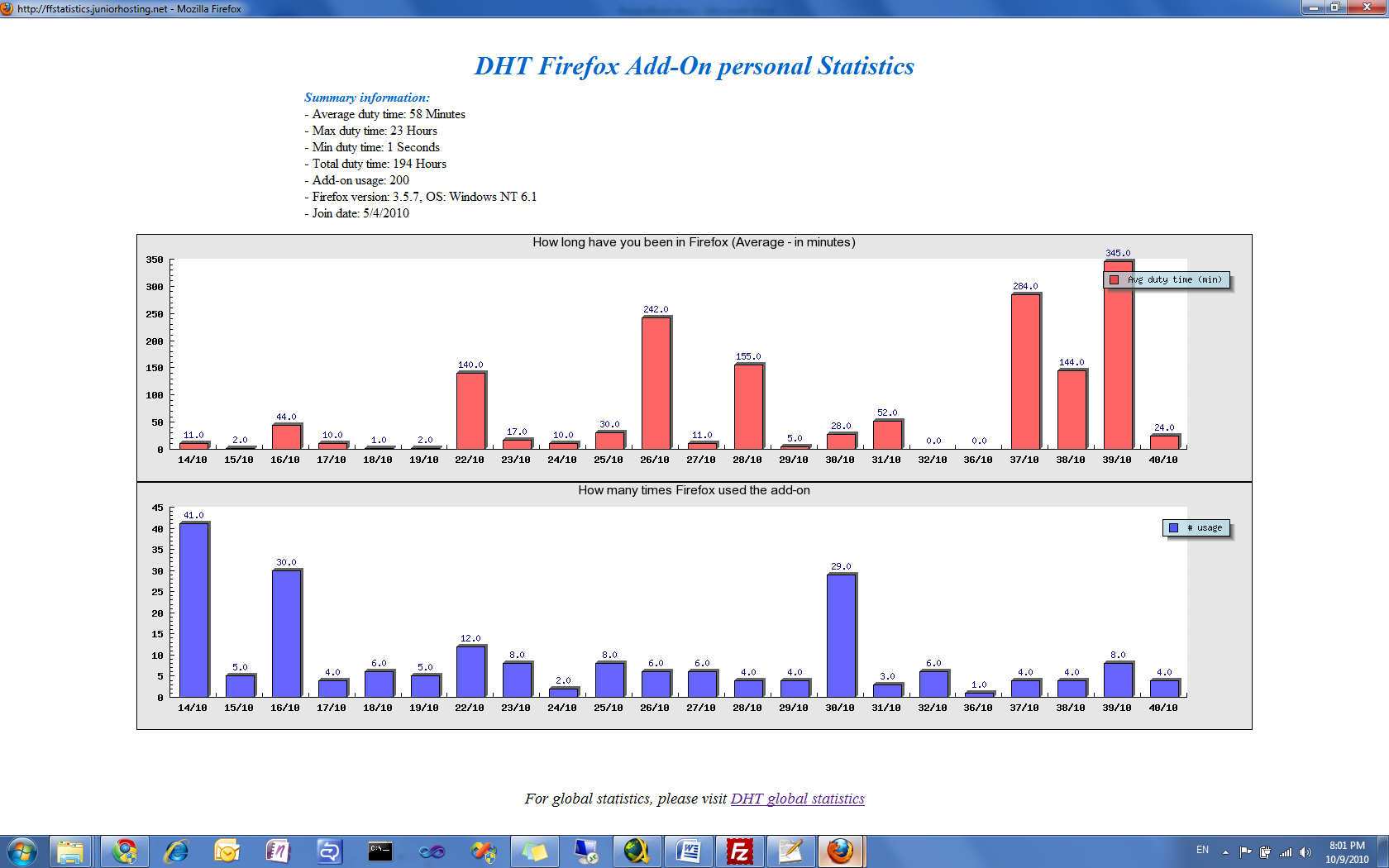
*Note: Why gathering and reporting logic resides on server (not client machines)?*

Gathering statistics and reporting logic resides on server machine in order to save consistent info.

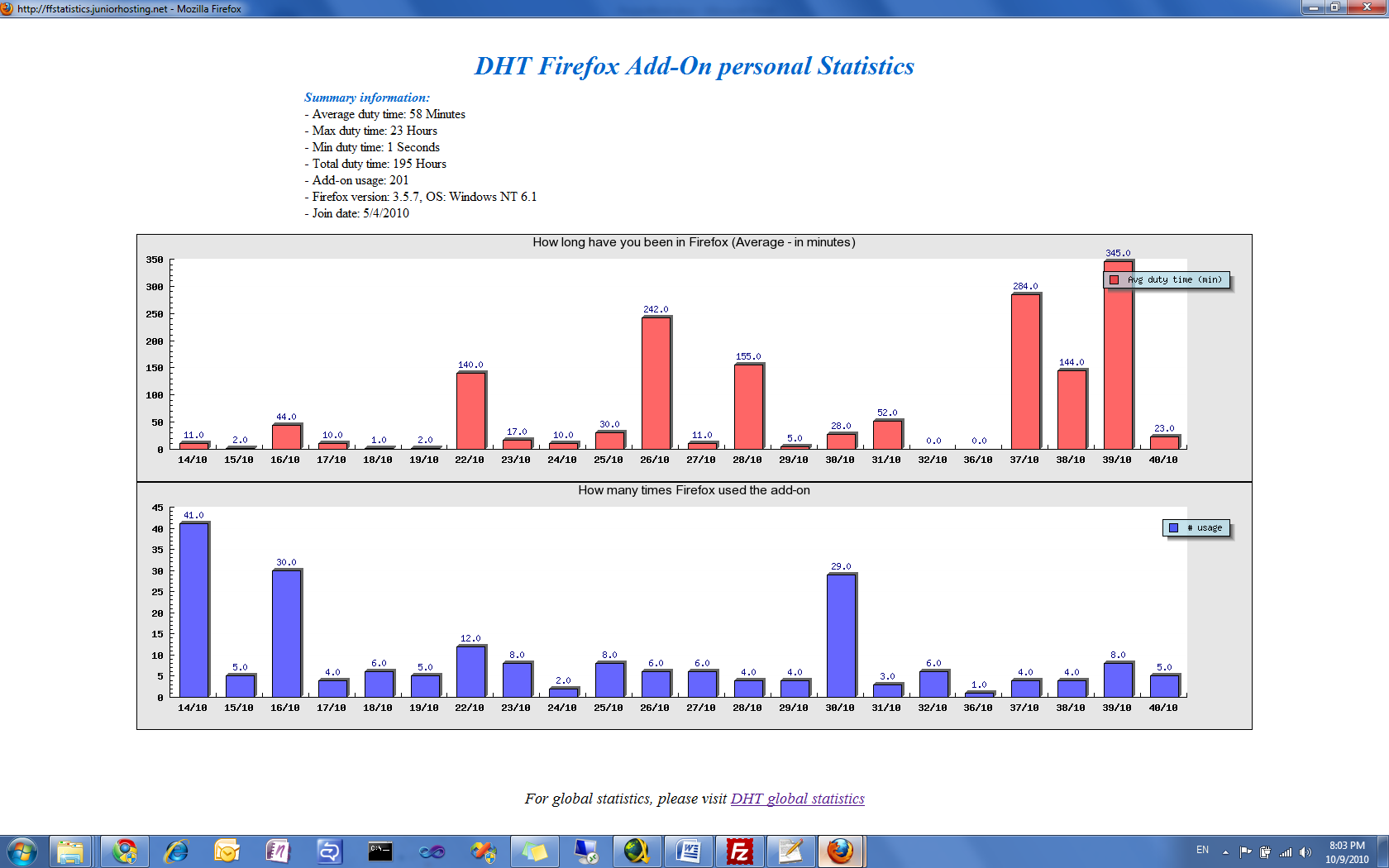
How? The problem in using client machine is the ***“Clock”***, which could be different between two different client machines. Using the server machine for gathering info allow us to use the same Clock since there is only one server machine.

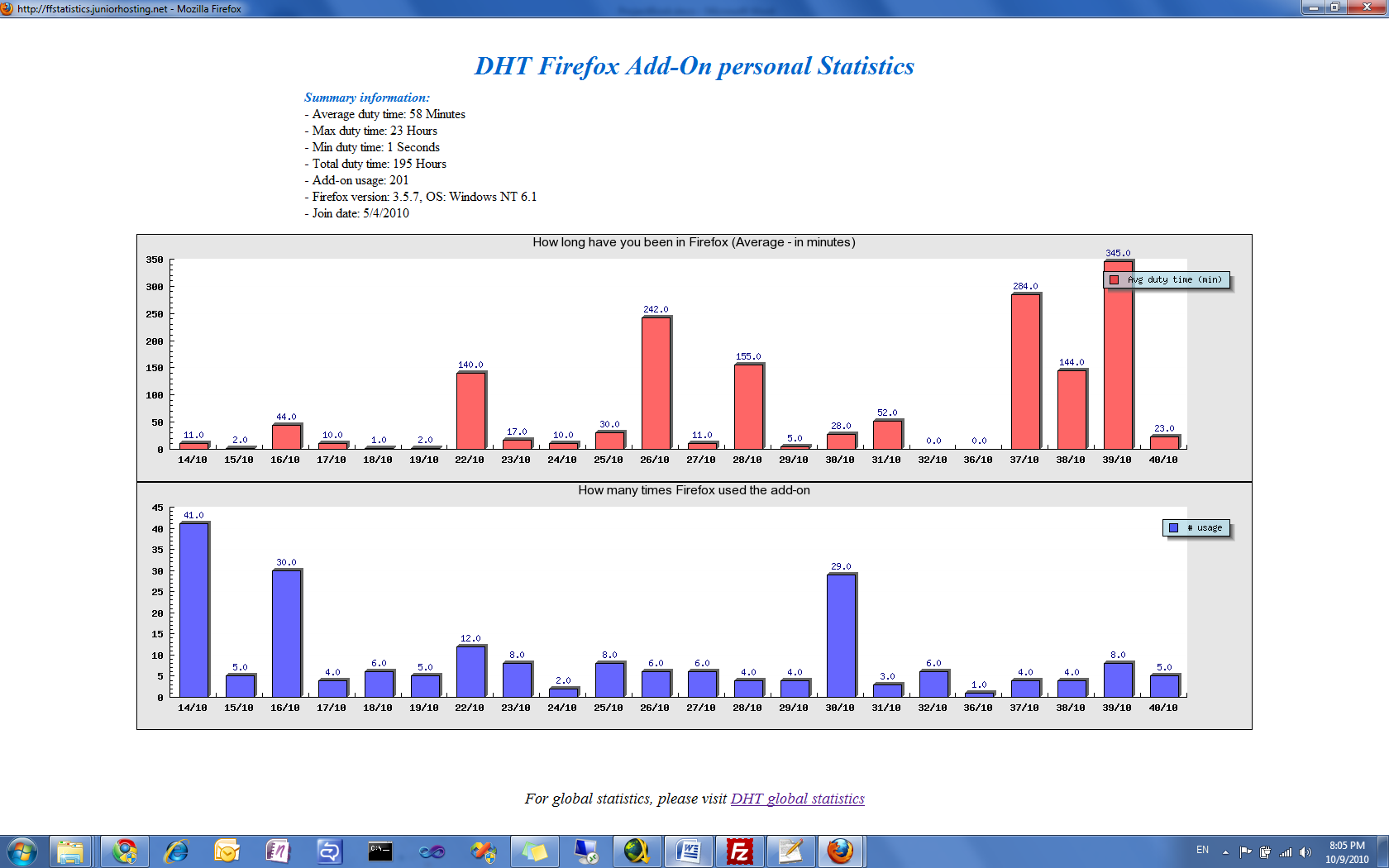
## Personal Info report (example)

Following are screenshots from personal info report:



*Personal summary info (example)*

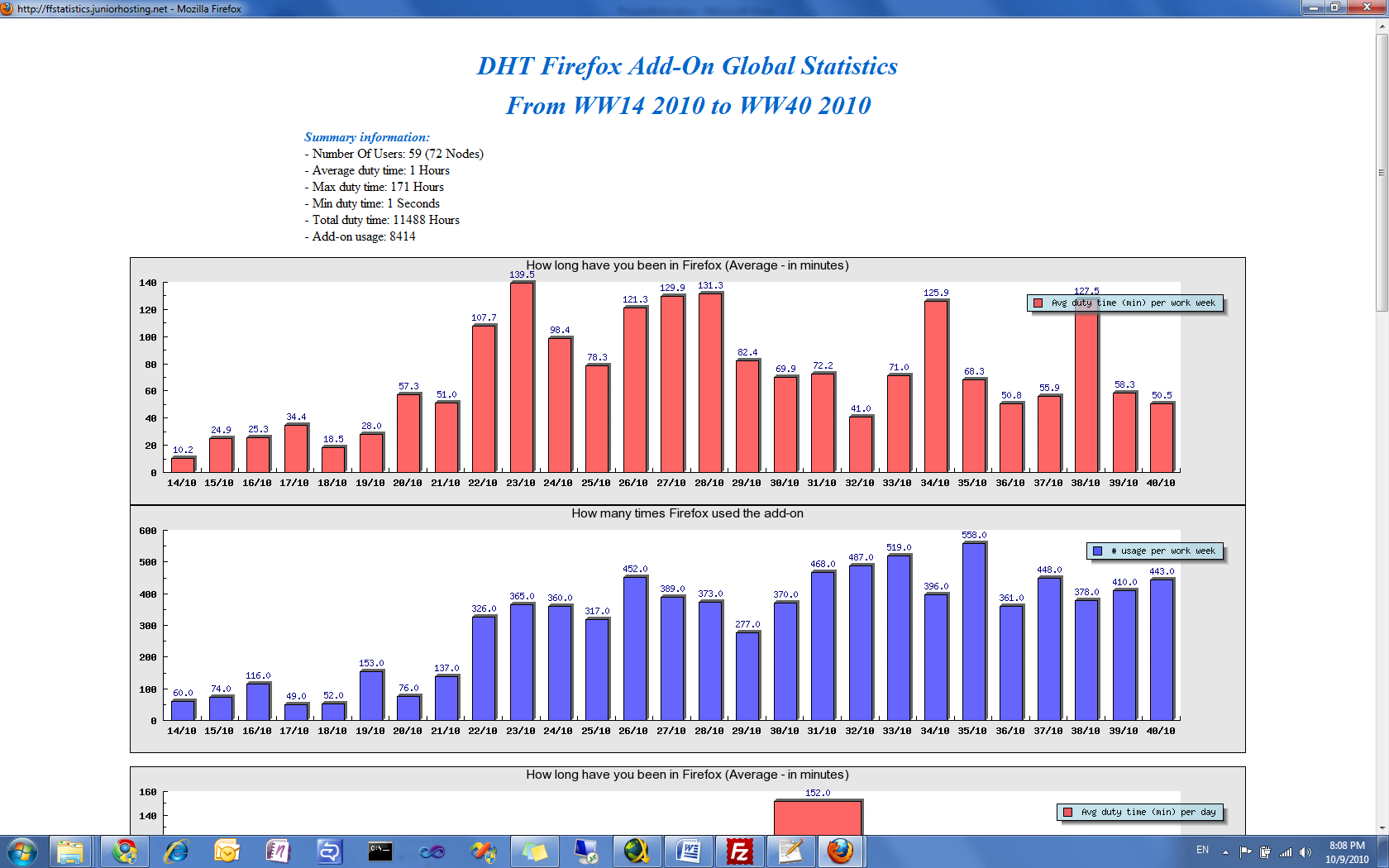
**

*Graph1: duty time per weeks (example)*

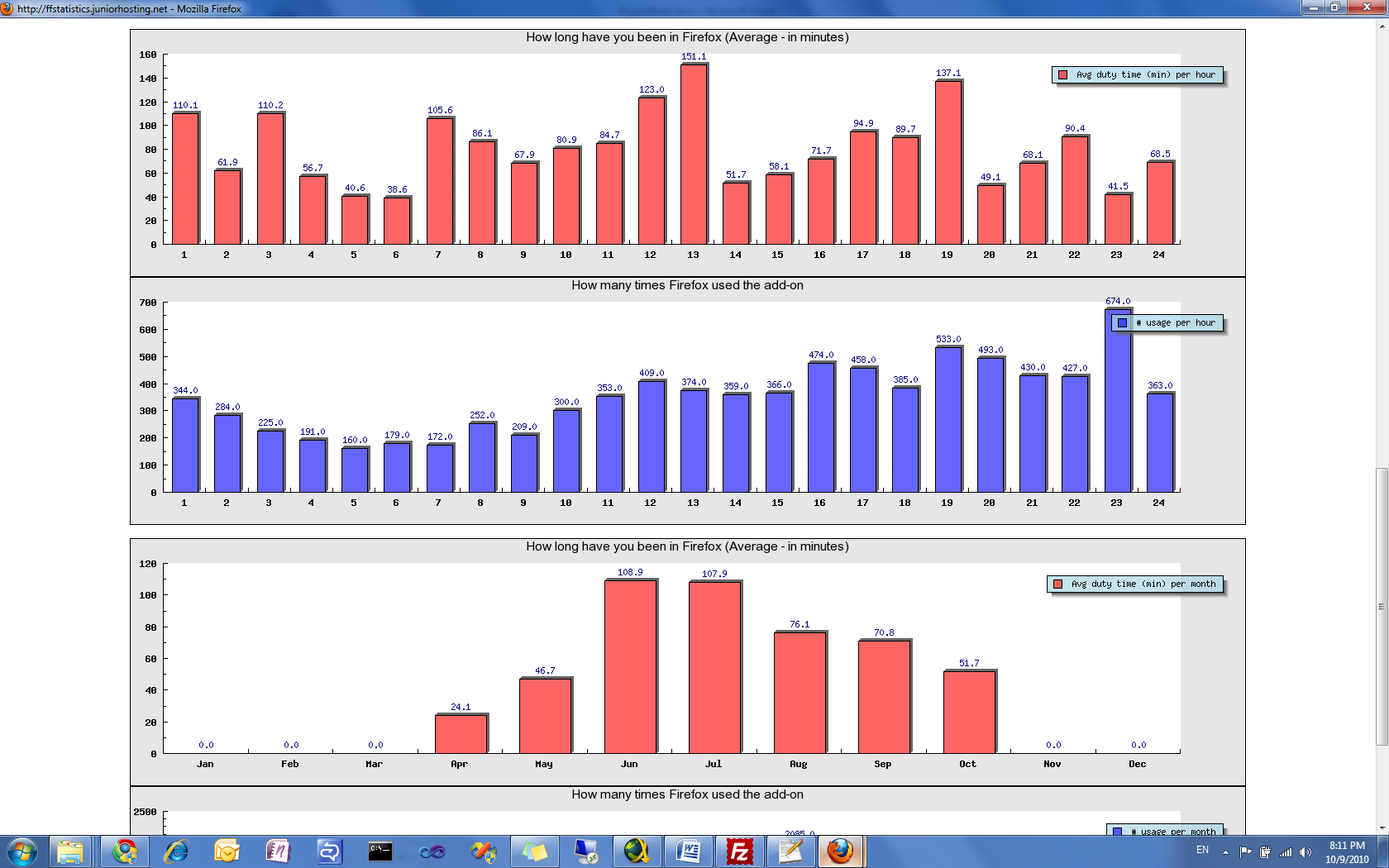
*Graph2: #time extension have been used per weeks (example)*

## Global Report (Example)

Following are screenshots from global report example:



*Global summary info (example)*



*Graph: total duty time and #times the extension used by all users by #hour*

***Same graphs are created (shows total duty time and #usage) by: Weeks\Days\Months***

# Analysis

## Known probability distributions

### Poisson distribution [6]

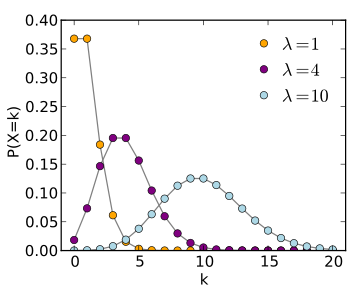
The Poisson distribution is a probability distribution that usually models the probability of the number of events occurring in a fixed period of time under the assumption that these events occur independently of the time since the last event occurrence and in a known average rate.

If λ is the expected number of occurrences in the given time-interval, then the probability mass function of the Poisson distribution for the variable k, which is the exact number of occurrences in that time-interval, is:

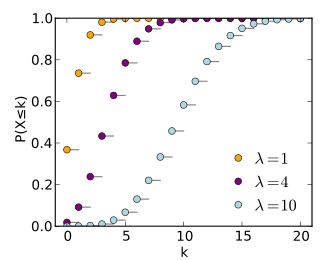
f(k; \lambda)=\frac{\lambda^k e^{-\lambda}}{k!},\,\!

Examples for situations where Poisson distribution presents a good model:

* Number of clients in a supermarket queue waiting some amount of time during a given time-interval
* End of life for products, i.e. number of products to break down during a given time-interval.



*Examples for the Poisson distribution probability mass function for different λ’s*



*Examples for the Poisson cumulative distribution function for different λ’s*

### Exponential Distribution [5]

While Poisson distribution models probability of the number of events occurring in a fixed period of time, Exponential distribution describes the time between events in a Poisson process. If λ is the expected number of occurrences in the given time-interval as described in Poisson distribution, the density function is described as following:



Example for situations where Exponential distribution presents a good model:

* Average time for supermarket client waiting in queue to come close one more step to his turn, means: average time spends supermarket cashier to finish his work with the waiting client.



*Examples for the Exponential distribution probability mass function for different λ’s*



*Examples for the Exponential cumulative distribution function for different λ’s*

## Goal

In order to try answering the main question of the project, we need to maximize statistics usage. The optimal approach is trying to create an analytic distribution function that fits for 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 analytic function is good?*

If we derive an analytic function from our statistics, it would be easy to analyze the function rather analyzing the statistics gathered especially if the function suits a known distribution functions such as Poisson or Exponential such as: finding average, maximum and minimum, draw sequential functions easy, find height cut for 2 dimensional functions and many others.

In the following chapters we’ll show analysis process done try after try in order to reach the final analysis approach which provides the most relevant information to answer project’s question.

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

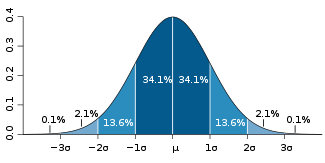
*What is standard deviation? [7]*

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 it’s defined as:



This means:

* Being far from the mean ±ơ assures that of the measurements to be in this interval.
* Being far from the mean ±2ơ assures that more of the measurements to be in this interval.
* Being far from the mean ±3ơ assures that more of the measurements to be in this interval.



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

*‘Large’* SD lowers the confidence level of predicting (error rate) the duty time of the next user. 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)***

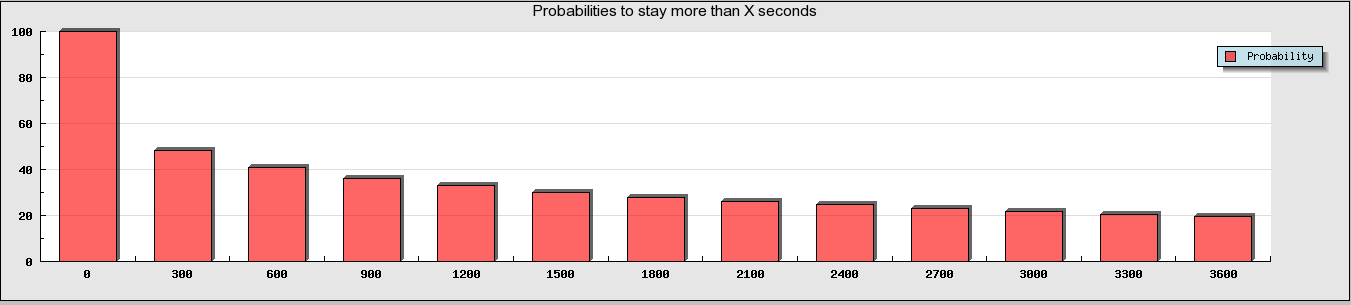
As we can see, SD is greater than average order, which means that SD is ‘Large’ and is not informative since prediction level is low. Therefore, Try 1: relying on the mean time is unconfident because of the large SD.

## Try 2: Choose known probability distribution function to fit for the statistics

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

Example:

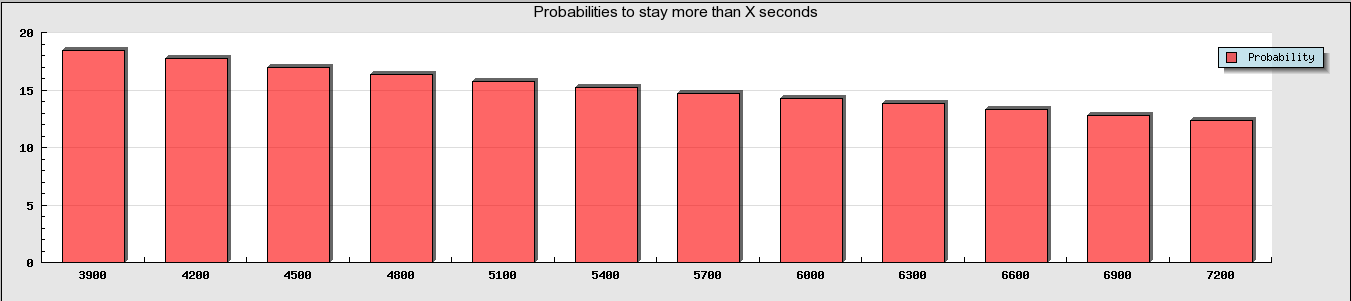


*The plot of the inverse accumulative distribution for the statistics we gathered, between 0 and 1 hour with an offset of 5 minutes.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X | 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 |
| Probability (out of 100%) 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.

## Final Try:

*What we really need from the statistics?*

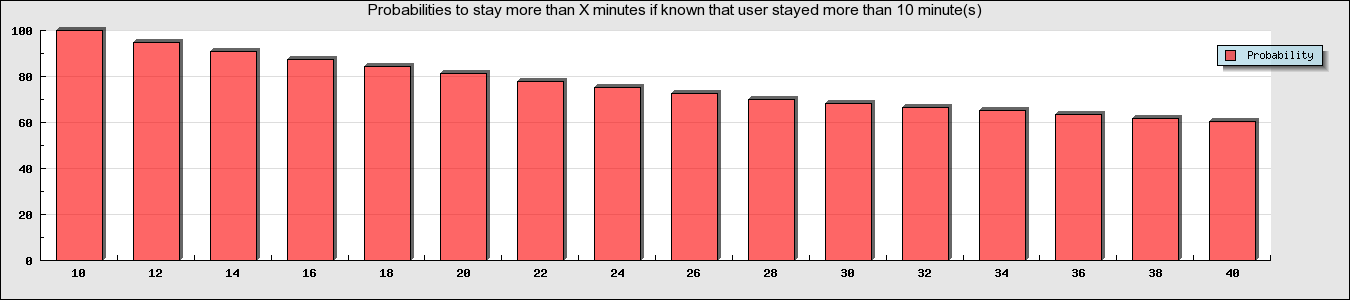
DHT is a dynamic protocol for distributing information across nodes. Predicting duty time for node\user that enters the system is the most important criterion for deciding what to do with it. Therefore, what we really need from the statistics is: *predicting duty time for a user that entered the system*, however, inverse accumulative probability of users with greater duty time than T helps deciding how much 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:



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (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 |

*The plot of the conditional probability (explained above) given that a node entered FF for 10 minutes for Xend between 10 minutes and 40 minutes with an offset of 2 minutes*

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

For instance, if a node staid for 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 and enables the DHT to give this node heavier data.

## 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 graph networks with several levels of data weights. Such that, the more confident we are that a node will stay ‘long enough’ (relative to the data weight) we will attach him to a graph 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. The arising question is therefore, how long should DHT wait before attaching a node to a 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 both the lightest DHT use, that will provide an indication of the maximal accepted and a maximal acceptable probability of that will provide an indication of the minimal accepted (which yields minimal accepted ).

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 graph network in a way that it considers a nominal time a node should stay online in order to be attached to the graph network. If this nominal time is not achievable under the minimal accepted probability then the DHT should choose not to attach the node to the graph network, and if the probability (to stay online for that nominal time) is higher than this minimal probability, the DHT will choose to attach the node to the graph 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.

and combined give an upper limit on the weight of lightest graph network. From assuming a minimal accepted probability of the user will “most probably” stay online for another time. This , considering an average transfer speed of the network should be enough time to transfer the largest piece of data in the lightest graph network (or otherwise there’s no sense in attaching **any** node in the first place). At this point, the DHT implementer decided the following aspects of our proposed DHT implementation:

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

XXX (ADD SKETCH - hayoub)

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. If the next checkpoint is “large enough” for the next data weight level graph network then the DHT will decide to attach the node to it, otherwise, won’t.

Example:

XXX

# Future project suggestions

XXX

# 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

XXXX