Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет «Высшая школа экономики» Факультет информатики, математики и компьютерных наук

Образовательная программа

«Интеллектуальный анализ данных»

Соколов Артем Сергеевич

Апробация модели децентрализованной системы с целью исследования возможностей распределенного полнотекстового поиска в реальных условиях

**Выпускная квалификационная работа по направлению Прикладная математика и информатика магистранта группы №15 МАГ ПМИ (магистерская программа «Прикладная математика и информатика»)**

Рецензент Научный руководитель

кандидат технических наук Старший преподаватель ПМИ

О. Р. Набиуллин А. А. Пономаренко

Нижний Новгород 2017

# Content

[**Content**](#_41a34d7c1wx1) **1**

[**1. Introduction**](#_le1met69v6n5) **3**

[1.1. Overview of related works](#_8e23bstahmvp) 5

[1.2. Goals](#_tpkdtd2i605l) 8

[**2. Technical details and problem solving**](#_7hz5jtyhpy6h) **11**

[2.1. BitTorrent Library](#_1mueittx5hzv) 11

[2.2. BEP](#_zcp6yz78rd01) 12

[2.3. Metric](#_xqx2xol9uaye) 13

[2.5. Network and node startup](#_838m12nxqqmz) 17

[2.6. Approximation problem](#_c8w9uybw3kxf) 20

[2.7. Conclusion](#_5ihcubjux48h) 20

[**3. Experiments and results**](#_76jis3t72ko9) **25**

[3.1. Data preparing](#_w18ypuo6fkhj) 25

[3.2. Test recall value](#_c5eiuuz2hed8) 25

[**4. Problem discussion**](#_a5qm8puni9ii) **28**

[4.1. Synchronization](#_d78e38pj7ntw) 28

[4.2. Security](#_imds8wfcj7qo) 30

[**Conclusion**](#_o9oh7eyktj5n) **36**

[**References**](#_mz9yz4m89gv2) **39**

[**Appendix A**](#_n8b8h3vmomzw) **41**

[1. Overview](#_oenj0yvv5tpl) 42

[2. Routing Table](#_xbemn9gaf24a) 44

[3. Term Frequency Vector](#_x3v6phd8fmzk) 45

[4. Torrent File Extensions](#_yq0ngwpmu6oi) 46

[5. RPC Protocol](#_itvvjb9wrj0e) 47

[6. NSW Queries](#_ylsntvl9nvir) 47

[6.1. Contact Encoding](#_gs2xw33fb15x) 47

[6.2. PING](#_xpc6toca93y2) 48

[6.3. GET\_FRIENDS](#_noqv6prymmco) 49

[6.4. ADD\_FRIEND](#_lrven3445287) 51

[6.5. SEARCH\_QUERY](#_43fbr9qdg264) 52

[7. Security](#_f9djm4rrbn02) 54

# 1. Introduction

Decentralized system research is on the cutting edge of the science of modern data analysis. Recently, due to the growth of IoT, it is becoming more and more relevant.

The advantages of such systems are known very well: damage resistance, scalability, and anonymity. The distributed hash table protocol “Kademlia” and blockchain technology are examples of decentralized systems that are already becoming part of the lives of ordinary people. Bitcoin as a cryptocurrency and a digital payment system showed that a distributed system can be open and very protected at the same time. The potential applications are endless and research will open more new areas. In the future, such services as web search can be implemented as a fully decentralized network as well. The current solution searches for information in a dump of all pages which is refreshed periodically. If all web servers are joined in the special network, pages can be found on the fly right after creation.

The theoretical possibility of the creation of that described in papers [1],[7] and a suggested special decentralized data structure is a navigable small world. Nodes in the network keep links to other nodes picked by a special approach. The authors considered that if this network is built, user can run text search from any node and search complexity will be logarithmical on average. They have provided some tests for proving on the easiest model search complexity. But there were no papers that described how this network can work internally. Because the “mechanism” for keeping system navigable properties during the network life is not obvious. The model from that papers is an approximation of a Delaunay graph and very likely specific routines should be implemented for saving properties for refreshing internal network information, detecting died elements of the network and replacing them.

Quite possibly, the following situation when text searching will not work appropriately during the active life of the network when some nodes die and new nodes are joined into the network.

In this work, we want to try to build a navigable small world system much closer to real systems for understanding the complexity of maintenance of the network, and find out problems to give guidance to further research for this kind of distributed systems. We will use the term Navigable Small World (NSW), meaning small world based on text distance measurement.

## 

## 

## 1.1. Overview of related works

Kademlia [2] is a Distributed Hash Table (DHT) that used by the popular peer-to-peer network BitTorrent for searching peers which store parts of the file. The protocol keeps search complexity while a number of peers are being changed continuously.

The [3] following is a description of the protocol:

“Kademlia uses a "distance" calculation between two nodes. This distance is computed as the exclusive-or (XOR) of the two node IDs, taking the result as an integer number. Keys and Node IDs have the same format and length, so distance can be calculated from them in exactly the same way. The node ID is typically a large random number that is chosen with the goal of being unique for a particular node. It can and does happen that geographically widely separated nodes—from Germany and Australia, for instance—can be "neighbors" if they have chosen similar random node IDs.

The exclusive or was chosen because it acts as a distance function between all the node IDs. Specifically:

* the distance between a node and itself is zero
* it is symmetric: the "distances" calculated from A to B and from B to A are the same
* it follows the triangle inequality: given A, B and C are vertices (points) of a triangle, then the distance from A to B is shorter than (or equal to) the sum of the distance from A to C and the distance from C to B.

These three conditions are enough to ensure that the exclusive or captures all of the essential, important features of a "real" distance function while being cheap and simple to calculate.

Each Kademlia search iteration comes one bit closer to the target. A basic Kademlia network with 2n nodes will only take N steps (in the worst case) to find that node.

Kademlia as DHT in common is the just extension for searching information about available peers, files and parts of files by file ID. The DHT mechanism calculates the info hash based on the data file. This hash is a unique file ID. A search is being run for the exact matching ID. This approach breaks internal semantics of the data. Info hash hides them and makes impossible searching for data internals. First of all, a user searches for unique ID on a central index that located on the WEB site (torrent tracker). This is the bottleneck of BitTorrent protocol which makes it vulnerable to censorship.

However, the NSW has some similarities with Kademlia and at least can try to inherit some of them:

* Resilience for attempts to disrupt the entire network
* Censorship resistance
* Obscuring what a client is searching for
* Obscuring who published a particular entry

The chord network suggests a successful implementation evolved from the idea of building a small world model described by Watts, D. J., and Strogatz, S. H[11]. But again, the network uses hashed keys for ordering and navigating. This protocol has some improvements which are used in some application. The “pure” version is not being used.

One more implementation of a similar peer-to-peer network that we should mention is Freenet. The goal of the network is censorship resistance.

“The Freenet file sharing network stores documents and allows them to be retrieved later by an associated key, as is now possible with protocols such as HTTP. The network is designed to be highly survivable. The system has no central servers and is not subject to the control of any one individual or organization, including the designers of Freenet. Information stored on Freenet is distributed around the network and stored on several different nodes. Encryption of data and relaying of requests makes it difficult to determine who inserted content into Freenet, who requested that content, or where the content was stored. This protects the anonymity of participants, and also makes it very difficult to censor specific content. Content is stored encrypted, making it difficult for even the operator of a node to determine what is stored on that node”[4].

Freenet is being built on the principle of small world networks. By connecting to nodes of people you already know, and the people you know in turn connect to people they know, one should be able to reach all nodes in a Freenet network. Thus, each node knows only about some number of other nodes that it can reach directly, but any node can be a neighbor to any other; no hierarchy or other structure is intended. Each message is routed through the network by passing from neighbor to neighbor until it reaches its destination. As each node passes a message to a neighbor, it does not know whether the neighbor will forward the message to another node, or is the final destination or original source of the message. This is intended to protect the anonymity of users and publishers.

This model can be used as anonymous file storage but searching through this network is impossible. It does not have any navigable properties and time of query, especially in the beginning of using, can take a long time.

A review of the existing literature shows that there are no precedents of created working NSW-similar systems. Existing implementations use quite different algorithms even they are based on small world principles. The main difference is that observed models use linear keys space and provide full matching search. They are not applicable for text query for data content (We don’t include here models which use broadcast queries for text searching because it is not a scalable approach). It means that the theme of this paper is relevant and makes sense.

All observed models have distributed models called peer-2-peer. It is quite logical if we try to implement as a p2p system too. As said above Kademlia is a DHT protocol used by BitTorrent for quick searching of file parts. We can create a protocol for searching .torrent files by its content. In a common .torrent file is a file that contains information about files and folders to be distributed. Also, it can include a field called “Comment” with a text description of the data.

Each torrent file is a vertex. It follows that a separate computer owns N points on the graph if it keeps N .torrent files. Our module can join .torrent files in the NSW network and provide an API for further search queries through the NSW of .torrent files.

## 

## 1.2. Goals

Papers [1], [7] theoretically showed that we can unite objects with some text description in a network to run quick a text search through it. The open questions are whether such a network can successfully build and live.. Authors made a trivial model to get experimental material but it does not have even simple synchronization messages to detect lost nodes. The existing implementation is not able to prove model viability. The papers avoided technical details. A number of problems should be resolved before network begin to live stably.

A possible successful implementation means that model researching goes in the correct direction and has insights to be used in modern technologies. An implemented model helps to identify disadvantages and problem places and possibly prompts the ways for improving.

On the other hand, if the implementation fails due to the complexity, it will show that papers should be reworked because an idea that discussed and described there yields a useful perspective only if it can be practically applied.

For the BitTorrent protocol, each intended enhancement is called an extension proposal and should go according to a certain process. We are going to prepare a draft document called BEP (BitTorrent Enhancement Proposal) for the feature and suggest them for editors from the BitTorrent.org community. If the BEP community approves, they assign the BEP a number, labels it as Standards Track, Informational, or Process, gives it status "Draft," and checks-in the initial draft of the BEP to the repository. The BEP editor will not unreasonably deny a BEP. Reasons for denying BEP status include duplication of effort, being technically unsound, not providing proper motivation or not addressing backward compatibility.

Standards Track BEPs consist of two parts: a design document, and an implementation. The reference implementation need not be complete when a BEP is submitted to the editors. However, Standards Track BEPs must include an implementation in at least one BitTorrent client with publicly available code before it can be considered final [6].

Thus, goals for our global work can be formulated in the following way:

1. Try to implement joining of .torrent files into an NSW network based on text similarity measurement as the extension of BitTorrent protocol for any bittorrent library.
2. Write the BEP (extension proposal).
3. Find out practical problems of the NSW network described in [7] and propose possible improvements.
4. Provide analysis of critical problems if they exist.

We understood that implementation, debugging, and problem-solving can take a lot of time. We determined 1-4 to be the minimum which we should achieve in this paper. If we will have time we would like to include the next tasks:

1. Do measurement tests to show model abilities with ~1000 nodes.
2. Investigate search complexity during the usual life of the network when nodes join and die randomly.

These are considered the maximum goals for the current paper.

# 2. Technical details and problem solving

This chapter is dedicated to a set of technical solutions and significant problems that appeared during analysing and implementation of Navigable Small World module for the BitTorrent library. The described ideal model in [7] often does not describe details, without which implementation is impossible. We are showing how and why we chose one or other approach for found problems.

## 2.1. BitTorrent Library

We were looking for BitTorrent library, where we would like to introduce full-text searching code and make a test with a BitTorrent client linked with this library.

We had the following criteria when choosing the BitTorrent library:

* The code is open source with the BSD license if possible.
* It is not mandatory to have very popular BitTorrent clients which use the current library.
* The library should be live and currently supported.
* The library should be well documented.
* The library should be in C/C++

We have chosen libTorrent from http://www.libtorrent.org/. This library satisfies all our requirements above. Besides,the library has more advantages such as:

* Good quality written code and code style
* Developers of this library are participants of bittorent.org community
* The library has command line torrent client for testing and makes it easy to check our changes and fulfill our experiments.

## 2.2. BEP

BEP stands for BitTorrent Enhancement Proposal. We need for it for official discussion by a community and for possible inclusion of an NSW module into the BitTorrent library.

We have published our idea description in the bittorent.org GitHub site. We got critical feedback and skepticism. But we received a lot of information clarifying what features of BEP we should specify in the future.

An example of one opinion:

*“Anyway, in my opinion a BEP would need to contain the following so that others could implement it in principle:*

* *the network protocol*
* *the algorithms for searching + example calculations where appropriate*
* *privacy analysis*
* *defenses against various malicious actors or at least an analysis where its weaknesses are*

*And that would just be the starting point for others to provide input, e.g. to improve privacy or add more defenses.”*

The BEP does not include a security specification due to the complexity of the problem. It requires additional investigation. We just have used feedback from the community as the basis for BEP writing.

As we have our own main goals for current work, we have taken into account points that were raised by the community, and have split the questions into groups for current and future implementations. In this paper, the following set of problems is mandatory to be resolved or described:

* Protocol and network-related questions
* Algorithms

Meanwhile, these issues must be resolved in the case of introducing the NSW module into BitTorrent. This paper offers some analysis in the “Problem discussion” chapter.

Thus the second group is:

* Privacy analysis
* Security-related questions

Code implementation caused continuous BEP changes and the current version corresponds to an NSW module. The full text of the BEP is published in Appendix A.

## 2.3. Metric

The metric used for the distance measurement calculation is restricted by the our using case. We cannot use existing dictionaries for text normalization and sophisticated known algorithms based on machine learning because in practice it requires the use of third party libraries. Not each BitTorrent client will use it due to license compatibility issues.

BitTorrent is used for a lot of countries with different languages and the algorithm should be valid for all of them at the same time. The writing of such a powerful library most likely will take significant time and does not make sense. The solution should work “in the box”.

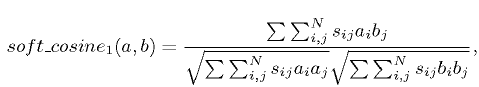
Paper [18] proposes to use simple cosine similarity for distance measurement between texts. The algorithm does not take into account order, synonyms, etc. This lightweight library is useful for calculating and works with many languages.

The description of the .torrent file likely should not be long. They often can be very similar. Possibly, words in descriptions may have 1-2 symbol differences (game - games) and for simple cosine similarity, this means two independent words. It is not quite intuitive and may give bad similarity measurements for closely related words. Possibly we can expect better results with short descriptions of a .torrent file if we take into account differences between words.

Natural language processing suggested the special procedure called “stemming” to take into account the similarity between words. From our point of view, a good decision is to try to replace cosine similarity with a soft cosine measurement. The algorithm calculates the distance between two words by, for instance, using Levenshtein distance [10].

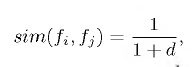
A dictionary with a small number of words should not have an impact on CPU or memory consumption. Another advantage that algorithms with the subquadratic complexity exist [9].

Authors of [9] proposed for example the following formula for comparing two text vectors:



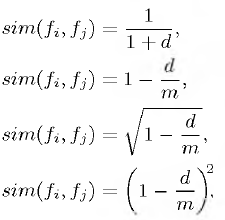
where 

Based on experiments made in [10] they recommended to use the following function for similarity calculation:



where  .

Full lists of of possible formulas is:



Where *m* is the maximum possible Levenshtein distance for two strings of the same length as the two given ones, which is the length of the longer of the two strings.

As there are no the best formula for all cases this is reasonable to do additional research for the most applicable soft cosine measure for our model. This is outside the scope of this work. That is why we just provided analysis and decided to use original simple cosine measure.

2.4. Links

The authors of [7] describe two types of links (edges of a graph): short, which are the approximation of Delaunay graph, and long, for keeping logarithmic search complexity. We have decided to store it separately in our routing table. We have specified constant K for the definition of a maximum number of stored closest nodes. The task of each node when it joins to the network is to fill all K vacancies. When a new closest node appears and asks to be added into the X node’s friend list, the X node should accept it if the distance between them and new one is better than with previously added nodes. Another node from our list with a farther distance should be moved into the list of old friends. The size of this list is not limited to our implementation.

In another case, we decline a new node.

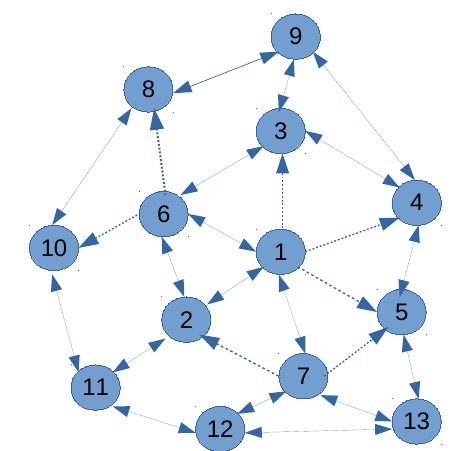


Figure1.

Figure 1 shows the situation when we have two types of edges. Bidirected are for closest nodes because both nodes keep each other in the first list. Single-directed showed “long” links and direction specifies who has been moved into the second kind of lists. For example, node 7 has moved node 2 and node 1 has moved node 5. But both of nodes 2 and 5 considers 7 and 1 as closest friends, respectively.

## 2.5. Network and node startup

The network building process has a start point when the are no running nodes at all.

The experience of Kademlia showed that we have to have pre-run nodes with some text. We called them “gateways.” The library includes code with IP address, ports and text description (for similarity measurement) of the nodes. In future, if NSW will be included as the extension of BitTorrent it makes sense to store some predefined addresses in a torrent file as Kademlia does.

Each node should have the entry point to run a bootstrapping routine and to be involved in the network. In our work, nodes use hard-coded addresses of “gateways.” It does matter to have several gateways with completely different text joined in the network. The starting node chooses the closest and begins the bootstrapping process from them.

Unfortunately, when two nodes join at the same time in one network area, they do not get information about each other and inconsistency in the network appears. That is why we have to run nodes one by one as shown in the Figure2.

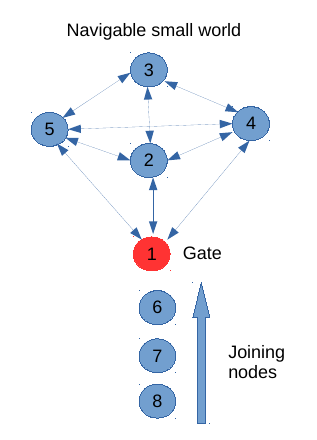


Figure2.

In chapter “Problem discussion” we will describe the possible solution for solving this problem.

One additional moment should be mention is that gateways should not have restrictions for the number of nearest friends. This helps to avoid situation when new node with description from significantly another area does not find closest friends and nobody includes it in own lists. If list is full and new node is not closer than nodes presented in the list it get rejected on the ADD\_FRIEND query. Gates should keep them till they create links with the network by new nodes.

## 2.6. Approximation problem

The theoretical graph is an approximation of the Delaunay graph - we cannot support great the number of links that the Delaunay model requires. It causes our model to have local extremum for some queries. To avoid finding local extremum during search queries or bootstrapping, we have to use heuristics.

According to our implemented model, each .torrent file is the node in the network. For a user query, we can use each file as the entry point and start N concurrent search routines from different parts of the network and merge results after that. Possibly they can find the same result if there are no local extremums, but otherwise, the approach can help to find our global extremum.

Another possible algorithm is extended BEP and includes messages for going to several random nodes in the network and run a search starting from there.

## 

## 2.7. Conclusion

We have implemented an NSW module for the BitTorrent protocol and extended a console test client for new abilities. The language used is C++ and elements of C++11 standard. The code is stored on a git repository[15]. Twenty-two classes were implemented and more than 6,500 lines of code were written. Our tests showed that the network builds successfully, nodes ping their nearest friends, and detect if they fall down. Nodes can be launched on different IPs but should not be separated by NAT.

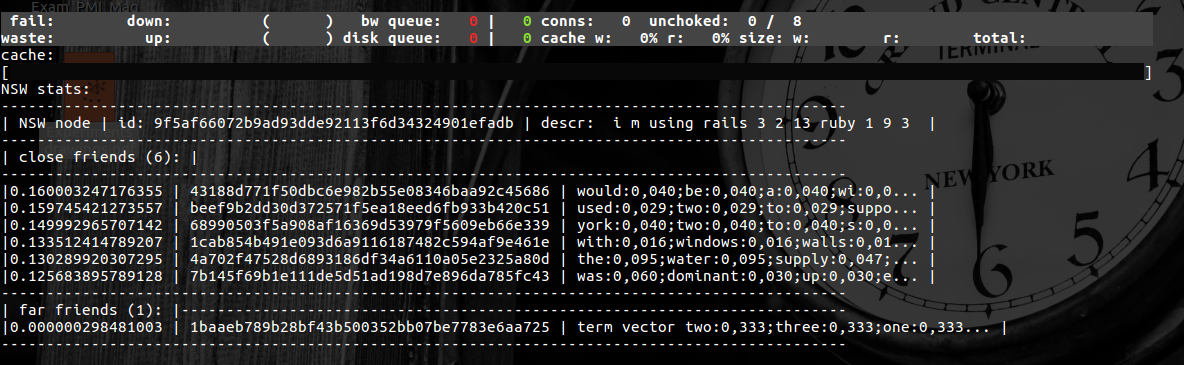
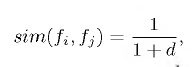


Figure3.

Figure 3 shows an example of a running node with connected close friends and old friends which are called “far friends” in the application.

The library supports simple cosine and soft cosine similarity metrics and changing of the closest friends list size.

We made quick research of soft cosine measure to understand the perspectives of this approach. We used the following formula for similarity:



We took thirteen nodes with the descriptions described the Table 1 (just first five results shown). They were launched in order mentioned in the first column. The NSW node with description “one two three” was in role of gate.

|  |  |
| --- | --- |
| 1 | one two three |
| 2 | i m using rails 3 2 13 ruby 1 9 3 and oracle 11g when saving a record in the db with a character like i get the error below activerecord statementinvalid encoding undefinedconversionerror u+00f1 from utf 8 to us ascii insert into omniauth\_users created\_at first\_name id last\_name status uid updated\_at values a1 a2 a3 a4 a5 a6 a7 app controllers user\_sessions\_controller rb 18 in create i tried to run this query to see the language used by oracle select userenv language from dual it returned american\_america al32utf8 these are my gems for oracle gem ruby oci8 gt 2 1 5 gem activerecord oracle\_enhanced adapter gt 1 4 2 gem ruby plsql gt 0 5 0 what should i do ruby-on-rails ruby-on-rails-3 oracle activerecord oracle |
| 3 | vertically aligned windows were set in black metal window frames and divided by black metal spandrels. The dominant feature was the centrally located entrance, which was highlighted by extending the central bay up |
| 4 | He began his career at GM as a financial analyst in 1986 and promoted in 1988 to GM's offices in New York City two10:created by |
| 5 | have been placed under protection since 1978 as cultural monuments.[2] The majority are still used, albeit nowadays their purpose is primarily to support rural conservation (the preservation of a historic cultural landscape) two |

Table1.

We created the network based on simple cosine measurement (Table 2) and rebuilt it with soft cosine approach (Table 3). Tables show similarities between nodes with according order IDs.

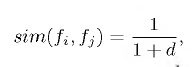
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| 1 |  | 0,0000002985 | 0,0000001667 | 0,1005039041 | 0,0936587376 |
| 2 | 0,0000002985 |  | 0,1256838958 | 0,1499929657 | 0,1597454213 |
| 3 | 0,0000001667 | 0,1256838958 |  | 0,1015677160 | 0,1419750485 |
| 4 | 0,1005039041 | 0,1499929657 | 0,1015677160 |  | 0,1129569124 |
| 5 | 0,0936587376 | 0,1597454213 | 0,1419750485 | 0,1129569124 |  |

Table2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| 1 |  | 0,9469796907 | 0,8496786081 | 0,8883228910 | 0,9322016956 |
| 2 | 0,9469796907 |  | 1,0000000000 | 1,0000000000 | 1,0000000000 |
| 3 | 0,8496786081 | 1,0000000000 |  | 1,0000000000 | 1,0000000000 |
| 4 | 0,8883228910 | 1,0000000000 | 1,0000000000 |  | 1,0000000000 |
| 5 | 0,9322016956 | 1,0000000000 | 1,0000000000 | 1,0000000000 |  |

Table3.

Simple looking is enough to say that distance based on soft cosine with measure with



is not applicable for our model as it gives very small variance. All descriptions look as very similar. As we said previously, distance based on soft cosine, potentially better than simple cosine, but required additional research At least each similarity suggested by [10] should be checked.

Each NSW node has the ability to send text search queries and dumps the result in the file. We have written a BEP that is totally consistent with the implemented model and can be shown for BitTorrent.org community.

However, the ability to replace dead friends is not implemented. This means that in the case of nodes falling down, the network may have holes. They can be repaired only if the network joins a similar node. The network cannot join several nodes concurrently. This is a much more complicated task for solving than the previous one and analysis of it is provided in the chapter “Problem discussion.”

Our minimum task is completed and the next step is to try to make tests which can show how the practical implementation is being correlated with results from [7].

# 

# 3. Experiments and results

## 3.1. Data preparing

As NSW node is a .torrent file and we should generate that file. We have used the “ctorrent” command line utility for generating. The file is metadata and we have provided the simple text file with one digit as an argument and text for the “Comment” attribute of .torrent. Text data are taken from [14]. This is set of articles from the StackOverflow resource. We have cut the first 1000 characters.

After that, we run BitTorrent test console clients with the generated file as the argument. Due to technical reasons, we launched one torrent file per client. This means that for getting 100 NSW nodes we launched 100 instances of test clients with different arguments.

## 3.2. Test recall value

The goal of this test is estimation of quality implemented search heuristics and proving of tests result got [7].

When the network is setup, we run a text query from a random node, receive the result, and compare if it is equal to our target. If yes, we have marked the result as positive. Recall in our experiments is equal to *positive queries/total number of queries*.

Calculated metrics is the average value of asked nodes before query finds the target. We have calculated this value separately for positive results when found target is expected.

Table 4 shows quite expected results, that recall is not great due to small number of closest friends and it decreasing when number.

Number of closest friends is 20. Distance is simple cosine similarity

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Network size | Queries | Recall | Calculated metrics (average) | Calculated metrics for successful cases (average) |
| 100 | 100 | 0.91 | 57.95 | 56.13 |
| 200 | 100 | 0.93 | 60.31 | 53.27 |
| 400 | 100 | 0.82 | 63.44 | 55.80 |
| 800 | 100 | 0.73 | 58.96 | 45.73 |
| 1200 | 100 | 0.81 | 60.76 | 47.43 |
| 2400 | 100 | 0.64 | 56.55 | 42.31 |
| 3200 | 100 | 0.54 | 43.21 | 29.81 |

Table4.

Our measurements showed not bad results for first version. We got high value of recall when launched search from one starting point. For that reason we have used greedy algorithm. Two last columns showed just calculated distances. If node provides all twenty closest friends, number of calculated metrics will be increased on six points. The columns show that in successful cases goal was closer than found local extremum for failed cases. This can be helpful information for build network and extremum decreasing algorithms improvement. For example, we can understand that fo nodes >2500, algorithm either finds target near him or does not find at all. (See Figure 5.)

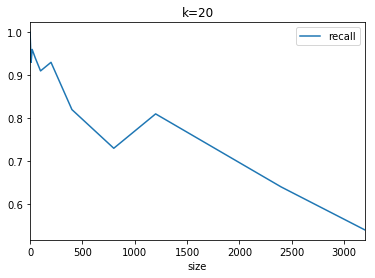


Figure4.

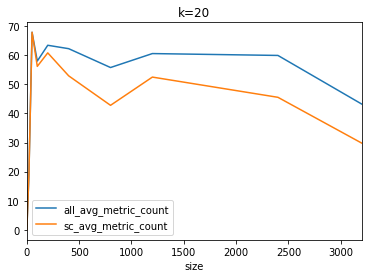


Figure5.

Recall can be increased by starting search from several NSW nodes simultaneously (which are handled by the same peer), or by improving searching heuristic, because authors of paper [7] published some improvements which can be applicable for our model.

# 4. Problem discussion

Tests and experiments showed that the NSW module could be and should be improved for as further research for including it as an official extension of the BitTorrent protocol. We have found out some significant problems in current protocol and library implementation. We suppose that these issues should be solved as a high priority in future papers. As we mentioned previously [7] does not discuss any technical details and that is why these problems are not obvious from the perspective of the original paper. But practicality requires issues to be resolved completely or partially.

## 4.1. Synchronization

Our implemented model starting takes very long time because we have to join nodes one to one. This caused by a message racing condition when a lot of nodes join simultaneously. NSW node X retrieves information about nearest friends of node Y and a moment after picture it has got can be not actual. Completed network will be built not correctly, a number of local extremums will appear and the Navigable Small World will lose its navigable properties.

The *gossip-based protocol* approach could be tried for solving this problem. The protocol name is caused by usual life interactions:

“The concept of *gossip* communication can be illustrated by the analogy of office workers spreading rumors. Let's say each hour the office workers congregate around the water cooler. Each employee pairs off with another, chosen at random, and shares the latest *gossip*. At the start of the day, Alice starts a new rumor: she comments to Bob that she believes that Charlie dyes his mustache. At the next meeting, Bob tells Dave, while Alice repeats the idea to Eve. After each water cooler rendezvous, the number of individuals who have heard the rumor roughly doubles (though this doesn't account for gossiping twice to the same person; perhaps Alice tries to tell the story to Frank, only to find that Frank already heard it from Dave). Computer systems typically implement this type of protocol with a form of random "peer selection": with a given frequency, each machine picks another machine at random and shares any hot rumors.”[13]

A *gossip-based* protocol relies on a periodic exchange of information between nodes. Such a period is called a cycle. Periodically, each node chooses a target from its closest friends, sends some information with picked one and processes the received information.

The [8] suggest the same idea and it is reasonable to use the approach for our model as well. Periodically each node should share its friend list with its nearest friends. Upon receipt of the list, the receiving node merges the list of friends received with its own closest node list to compose a new list of neighbors.

In this way, message exchanging provides removing of local extremums which can appear after concurrent node joining and ensures that navigable inconsistencies will not appear at all. Besides of it, gossip protocol helps quickly find out replacing nodes to replace dead ones. Described cases showed that protocol or similar analogs would be useful to have in the future in BEP to provide stable navigable properties and high value of recall.

Technically, existing PING messages could be used for this goal. The PING is implemented in our model for checking live nodes on the list. The next node for the ping is picked based on the time of the last ping. This provides a uniform and regular survey of all friends. Changes to the body of the ping message will be required but other things like timers and message handlers exist and require the minimum of impact.

## 

## 4.2. Security

Obviously such a public network should guarantee as much privacy as possible and have defenses against various malicious actors. The current implementation of the module provides neither the former nor the latter. Security in general is a big concern of this version of NSW. Inheritance of existing security mechanisms as the DDoS blocker, obfuscation, and encryption which Kademlia uses for the time being can help to resolve a number of security issues. But we are expecting that the network is open for each new node, and as NSW functionality is based on open-text data metrics, it causes the network to be very internally vulnerable to destruction. Some attack examples are as follows..

The malefactor creates nodes (.torrent files with descriptions) with inappropriate or malicious content. It means that content of the file which will be downloaded can mismatch with the .torrent file description maliciously. The network now is not able to detect and exclude such elements or at least protect from them another network.

For instance, for a potentially popular search target, we can easily generate a number of fake nodes with the reason of providing and promoting our content. It is a quite complicated task to filter out them for a search query. There are other similar problems when we fake nodes are generated to surround a target that we don’t want to be found.

Such attacks can partially or completely (depends on the size of NSW) block the network from working. Most likely without partially or complete solving these problems, the BEP has a small chance to be accepted as a draft version.

The described problems require significant research and this is out of the scope of the current paper. However, we would like to try to provide a concept of possible network guarding here. To start with, it makes sense to relax the problem. Try to solve it and return to original conditions.

In this way, we can consider that our network not fully decentralized and we have there exist some number of servers for goals of managing and maintaining the network. In that case we can try to introduce Direct Anonymous Attestation (DAA) algorithm with signature revocation lists (Intel(R) EPID as an example). For instance, gates which are used as entry points can also be in the role of mentioned servers.

“DAA is a digital signature algorithm supporting anonymity. Unlike traditional digital signature algorithms, DAA provides a common group public verification key associated with many (typically millions) of unique private signature keys. DAA was created so that a device could prove to an external party what kind of device it is (and optionally what software is running on the device) without needing to provide device identity, i.e., to prove you are an authentic member of a group without revealing which member. EPID enhances DAA by providing an additional utility of being able to revoke a private key given a signature created by that key, even if the key itself is still unknown.”[12]

Each node has part of its own private key. When it joins to the network, it gets the second part of the private key and the group certificate with the public key as part of the special server (this is a gate in our model). The group private key is stored in the cloud with access for gates only (see Figure4.). There is at least one group for all NSW nodes in the community; however, more groups can exist if it makes sense. Peers (owners of NSW nodes) and gates have signature revocation lists. In a typical use case, nodes may keep the signature revocation list locally and gates on the cloud as well. Revocations lists are used for saving the public keys of nodes that are suspected of malicious activity.

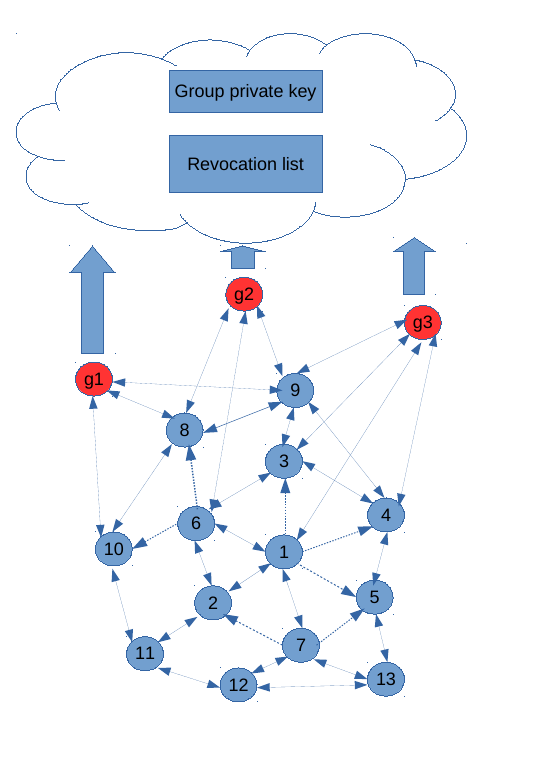


Figure4.

A *gossip-based* *protocol* relies on a periodic exchange of information between nodes. Such a period is called a cycle. Periodically, each node chooses a target from its closest friends, sends some information to the selected target and processes the received information.

The [8] suggest the same idea and it is reasonable to use the approach for our model as well. Periodically each node should share its friend list with its nearest friends. Upon receipt of the list, the receiving node merges the list of friends received with its own closest node list to compose a new list of neighbors.

In this way, message exchanging provides removing of local extremums which can appear after concurrent node joining and supervises that navigable inconsistencies will not appear at all. Besides of it, *gossip protocol* helps quickly find out replacing nodes to replace dead ones. Described cases showed that this protocol or a similar analogs would be useful to have in the future in BEP to provide stable navigable properties and a high value of recall.

Technically, existed PING messages could be used for this goal. Ping is implemented in our model for checking of live nodes from the list. The next node for the ping is picked by the last ping time condition. This provides a uniform and regular survey of all friends. Ping message body changes will be required but other things like timers and message handlers existed and require the minimum of impact.

Before handling each request (except PING), the node checks if another side is in the group. It may be done by the following way:

1. Node X sends GET\_FRIENDS request to Node Y
2. Node Y sends own revocation list to X
3. X calculates and sends proving for each line in list that this entry is not node X
4. If X proved that it is not revoked, node Y provides response for GET\_FRIENDS request

For protecting from large revocation lists, gates can run the REKEY command for the whole network to refresh all signatures and make revocation lists empty.

During text search, joining and other queries, the node asks the requestor to prove if he is good standing in the group of trusted nodes. Peers can exchange their revocation lists with each other with the help of, for example, the *gossip-based protocol* described before. The algorithm of adding new nodes which we have got from *gossip* should be not trivial. Possibly elements of voting should be presented there. By this way, malicious malefactors will be isolated and do not have the ability to message with other elements of the network.

Regardless, gates may have and maintain revocation lists too and NSW nodes or peers may send complaints about malicious nodes. The gate can decide to remove the public key from group and node will be removed from the NSW network. Again, the algorithm for making a decision should be not simple and should take into account several parameters.

However, there is a problem that a node can generate a new key and return back in the network. We could not catch it, but if the same behavior is repeated, it will cause the node’s isolation or removal from the network. Possibly malicious malefactors may try to send complaints about other normal nodes with the goal of bringing them into disrepute. This problem should be solved by the gate revocation algorithm.

This is just a concept and it requires detailed research. From our point of view, some modification of this idea possibly can be mapped for the original problem as well. The community will have the only ability to isolate the problem.

One more approach for increasing security would be to move the NSW network into DarkNet. This means that each node can communicate with nearest friends only. The same approach is being used by modified FreeNet. Node asks their friends for text search. They ask their own friends and so on. Requests and answers go through the sequence of nodes and the requestor connects with the target only on the last stage. This causes a significant increase of privacy, as during the request we cannot retrieve information about other nodes - only about our nearest friends, and about the target only if it exists and accept us.

# Conclusion

The NSW model is the structure with the high potential. It can be used in different spheres from the web to IoT. It is necessary to give a starting point by providing a successful implementation and showing the abilities of the model by tests and experiments. In this paper, the first step was attempted.

We have found out and described the practical problem regarding node synchronization for timely repairs of inconsistencies in the network after nodes leaving the network. For real life, this is mandatory to be solved because the network is living and a number of nodes are joining and leaving the NSW simultaneously.

Security is the second problem of this kind of network. If security questions are not solved, it causes significant restrictions on the area where the model can be used. Some protection approaches can be inherited from Kademlia, but NSW is also vulnerable to destruction from inside. There is no obvious protection and this is another direction for research. For both problems, we have suggested ideas for searching for a solution.

We have implemented NSW as the extension of BitTorrent protocol for searching a .torrent file by its description and put it into a git repository [15]. The language used is C++ and elements of C++11 standard. About 20 classes were implemented and greater than 6,500 lines of code were written. Our application works stably and allowed run measurements tests. We made experiments and proves navigable properties of our built model. Recall value we got is not so big but we used a simple model with one entry point. Ways for improving are quite obvious and we described them in this paper.

Unfortunately, we have not completely achieved our maximum goals. The current library implementation should be improved by repairing the routines for inconsistencies. After that stress tests should be done. The key idea is to understand how the recall is being changed when the network is not static - nodes leave it and enter it permanently. The results should show the direction of further research because stable dynamic decentralized systems have much more perspectives. NSW can use any other n-dimensional metric instead of text similarity measure. If the network can be stable with dynamic conditions it has good chances to be used in IoT.

We also have prepared the official documentation, BEP (BitTorrent Extension Proposal) and this fact allows us to continue the discussion with the bittorent.org community for including our module as an official extension to BitTorrent protocol.

Resume of our work for the current paper is the following:

1. Have implemented the NSW network module for .torrent file searching. About 7000 lines of code were written.
2. Have written BEP for further including our module in extension list of BitTorrent protocol
3. We have found out implementation problems of the NSW and part of them we solved in our code. Others we have described and suggested possible solutions.
4. We have provided analyzing of security problem - great problems of this kind of networks.
5. Have done tests which proved navigable properties of the network when a number of nodes are ~3000.

We consider next steps should be:

1. Implementation of mechanisms to repair inconsistencies by running bootstrap if nearest friends have left the network or died.
2. Implementation of *gossip-based protocols* for exchanging friends.
3. Stress testing of navigable properties for the live dynamic network.

From our point of view, theoretical research should be addressed regarding security and privacy problems.

# 

# 

# References

[1] Пономаренко. Организация быстрого поиска без индекса. 2013

[2] Peter Maymounkov, David Mazieres, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric", IPTPS 2002.

[3] Wikipedia. https://en.wikipedia.org/wiki/Kademlia

[4] Wikipedia. https://en.wikipedia.org/wiki/Freenet

[5] Wikipedia. https://en.wikipedia.org/wiki/BitTorrent

[6] BitTorrent. http://www.bittorrent.org/beps/bep\_0001.html

[7] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, Vladimir Krylov "Approximate nearest neighbor algorithm based on navigable small world graphs", 2013.

[8] O. Beaumont, A.-M. Kermarrec, L. Marchal, E. Riviere., VoroNet: A scalable Object Network Based on Voronoi Tessellations, Long Beach, US, 2007.

[9] Wikipedia. https://en.wikipedia.org/wiki/Cosine\_similarity#Soft\_cosine\_measure

[10] Grigori Sidorov, Alexander Gelbukh, Helena Gomez-Adorno, and David Pinto “Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model” Computacion y Sistem as Vol. 18, No. 3, 2014 pp. 4 9 1-504

[11] Watts, D. J. and Strogatz, S. H., Collective dynamics of ’small-world’ networks. Nature 393 (1998), 440–442.

[12] Wikipedia. https://en.wikipedia.org/wiki/Enhanced\_privacy\_ID

[13] Wikipedia. https://en.wikipedia.org/wiki/Gossip\_protocol

[14] https://cloud.mail.ru/public/3bwi/bFYHDN5S5

[15] https://github.com/artsokol/libtorrent, branch decentrilized\_text\_search

[16] Thomas Hardjono, Ned Smith, Alex (Sandy) Pentland Anonymous Identities for Permissioned Blockchains

[17] E. Brickell and J. Li, “Enhanced Privacy ID: a Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities,” IEEE Transactions on Dependable and Secure Computing, vol. 9, no. 3, pp. 345–360, 2012.

[18] Arasu A. et al. Searching the web //ACM Transactions on Internet Technology (TOIT). – 2001. – Т. 1. – №. 1. – С. 2-43.

# 

# 

# Appendix A

BitTorrent Enhancement Proposal

:BEP: n/a

:Title: Navigable Small World (NSW) Protocol

:Version:

:Last-Modified:

:Author: Alexander Ponomarenko <aponom84@gmail.com>, Artem Sokolov <artsokol87@gmail.com>

:Status:

:Type:

:Created: 12-May-2016

:Post-History: 12-May-2016: Initial version

BitTorrent may use a "navigable small world" extension (NSW) for joining .torrent files into a network for further navigation through them. As a result, a user can find a .torrent file with the closest text to the search query without using a modifyedcentral tracker. A text description is stored in the .torrent file. The protocol has similar routines as Kademlia [#Kademlia]\_ and is implemented over UDP.

For simplifying future possible implementation, we will use similar terminology and protocol technical details as Kademlia uses. A "peer" is a client/server listening on a TCP port that implements the BitTorrent protocol.

A peer is an owner of a number of NSW nodes. "NSW node" is a .torrent file interpretation with a text description. The field "Comments" may be used for this purpose, but in future it will be better to use a special field in a .torrent file that should be encrypted.

The "navigable small world" network is NSW nodes joined in a special way. BitTorrent clients keep .torrent files, NSW nodes, that are just points in different places in a global NSW network and each node has a connection with a small number (predefined constant) of other NSW nodes.

Asimilar idea underlies Freenet [#FreeNet]. But as Freenet uses Kademlia, it uses linear-ordered keys space. However, their expressiveness is naturally limited by the exact-match interface they provide. This makes it impossible to search through the data using a text search.

### 1. Overview

Each peer has a globally unique identifier IP and port ("Peer ID" in further). NSW nodes, as representation of .torrent files, use the .torrent infohash as ID. Two equal .torrent files that belong to different nodes and have the same ID are the same point in the NSW network. All NSW nodes use a "similarity metric" [#NavigationInSmallWorld]\_ between two NSW nodes for "closeness" calculating. A routing table is used for storing such information. Each NSW node keeps in the routing table just at least k nearest connections to the other nodes - in other words, a "friend list." The bootstrapping process stops after k nearest nodes are found or there are no more not-visited nodes in the network. After that, the list is changed during the life of the node. Experimental results showed that k is not big.

A text comparison is done by text term frequency vectors. A term freq vector is a set of words from any text with an associated frequency for each word. Similarity is a digit between 0 and 1 and can be computed as the cosine of the angle between term frequency vectors of two NSW nodes:

cos(alpha) = A\*B/(||A||\*||B||) where A and B are term vectors. Two closest term vectors have a similarity of ~1. Distance is 1 - similarity can be used as well.

In the above scheme, the network has the ability to run a quick search through itself.

The following algorithm is used when the user queries for some text string:

1) Term vector is computed for a query string.

2) A peer runs a query through all handled nodes.

3) A query with a term vector is sent by each NSW node to its friends.

A nodes asks its closest friend to provide information about its K friends in order of closeness to the query. The peers’ nodes are entry points for the searching algorithm. The .torrent file points are located most likely on different places of the small world grid - they are not neighbors. To begin a search from different places allow to avoid of retrieving local extremum as final result.

4) Queried node returns ip addresses, ports, IDs and similarity of nodes from routing table according to received term vector similarity.

5) The original small world node (or user query) iteratively ask nodes from

incoming messages.

6) The algorithm stops when it reaches a minimum in distance metric:

a NSW node whose friend list does not contain closer to the query node

than itself returns list of friends with itself on the top. It follows that

closest to query torrent file found. As we on each step have distances

we can find K closest nodes using algorithm from [#NavigationInSmallWorld]\_.

The following algorithm is used in bootstrapping routine when NSW node looks up for nearest other nodes:

1) Some special nodes (gateways) should be added manually into routing table. They can be specified in torrent file or passed as parameters when user application has run.

2) p2-p5 from previous algorithm

...

6) Request K nearest nodes for adding it in their routing table and provide their term vectors

7) Add node and its term vector in own routing table

### 

### 2. Routing Table

Each NSW node keeps routing table. As it said previously after bootstrapping routing table will include k closest friends only for each NSW node. But while node is active it can increase this number by not removing old connections when new node appears in network and asks closest friends for adding. It does matter to have separate list for nodes which becomes not in K closest after adding new one. Virtually, such nodes moves step by step in other part of navigable small world. Links with these nodes become relatively longer. This heuristic provides keeping of search complexity be equal log(n). After reaching of a certain constant number of nearest nodes, new adding routines should replace existing nodes with the lowest similarity.

The list of friends in the routing table are used as starting points for queries in the small world. Nodes from the routing table are returned in response to incoming queries from other nodes.

Initially routing table for each node should not be filled in by nodes from this peer. User application or torrent file provides "Gateway".

The following things can be used as they are in Kademlia:

1) Routing table stores "good" nodes only. "Bad" nodes peer of which fell down should be removed from routing tables and new friend should be found. After 15 minutes of inactivity, a peer (and all belonged nodes) becomes questionable. Node from friend list appears to be "bad" when it don't answer on multiple node messages. It causes questionable status for all NSW nodes with the same peer ID in friend lists in current peer.

2) If there are any questionable nodes they should be pinged. If such node fails to respond to a ping, it should be replaced by another one. It means that new searching process should be launched.

3) Kademlia uses "last changed" property for buckets. NSW node has the same property showing how old last changes made. This field is updated after each incoming message from/successful outgoing message to this node.

4)The routing table should be saved between invocations of the client software.

### 

### 3. Term Frequency Vector

Text description from Torrent file "Comment" field is presented as vector of frequencies for each word from "Comment". Description should be tokenized (spaces and punctuation signs should be removed). Frequency of each word appearances calculated. Frequency for a word "aword" is **number\_of\_aword\_occurences/total\_number\_of\_words**. Method does not take into consideration a words order. Comparing of two such vectors made by cos measure as known as "cosine similarity". Term vectors are being created when user application reads new torrent file or on startup.

The disadvantage of this approach is considering words in vectors as

independent or completely different. For example "film" and "films" are

not the same from cosine similarity point of view and thus they are

mapped to different dimensions of model.

We found this simple and easy for understanding approach enough for first version of network. But we are planning to replace this approach on "soft cosine" measure based on Levenshtein distance. This algorithm is similar calculated but it also takes into account possible terms relations.

We did not use stop words or stemming because it requires third party libraries linking with according API and language dictionaries. We would not like to involve third party libraries but we understand they are more powerful for text comparing and it makes sense originate separate BEP for this theme.

### 4. Torrent File Extensions

Term frequency vector is being built basing on text written in "Comment" field. It is enough for first version of protocol but it would be better to use special encrypted field that would be out of info hash calculating. This done for considering of two absolutely equal files with not equal descriptions as the same node on small world grid.

### 5. RPC Protocol

RPC mechanism in common is inherited from kademlia and uses bencoded dictionaries and has the following features:

- single packets for query/response

- no retry.

- three types: query, response, error.

- three queries: ping, get\_friends and add\_friend.

- using transaction ID

### 6. NSW Queries

All queries have an "q\_id" key and value containing the node ID of the

querying node and "r\_id" with node ID of recipient. All responses have

an "r\_id" key and value containing the node ID of the responding node.

The "\*\_id" is 20-byte string calculated as info hash of file.

#### 6.1. Contact Encoding

Contact information for sws nodes is encoded by the same way as in kademlia:

"Compact node info" is a 26-byte string where

- 20-byte Node ID in network byte order

- 4-byte IP address is in network byte order

- 2 byte port in network byte order

#### 6.2. PING

The message has only one difference from kademlia ping: as several nodes can be hidden behind one IP address ping message should specify both ID of querying node and node to be checked by ping.

Key "**q\_id**" - id of querying NSW node

Key "**r\_id**" - id of recipient/response NSW node

If node with "**r\_id**" exists, response message has repeated response id.

**arguments**: {"q\_id" : "<querying nodes id>", "r\_id" : "<queried nodes id>"}

**response**: {"r\_id" : "<querying nodes id>", "q\_id" : "<queried nodes id>"}

Example Packets

**ping Query** = {"t":"aa", "y":"q", "q":"ping", a":{"q\_id":"abcdefghij0123456789", "r\_id":"0123456789abcdefghij"}}

Spaces are just for easier reading:

bencoded = d 1:a d 4:r\_id 20:0123456789abcdefghij 4:q\_id 20:abcdefghij0123456789 e 1:q 4:ping 1:t 2:aa 1:y 1:q e

**ping Response** = {"t":"aa", "y":"r", "r": {"q\_id":"0123456789abcdefghij", "r\_id":"abcdefghij0123456789"}}

bencoded = d 1:r d 4:q\_id 20:0123456789abcdefghij 4:r\_id 20:abcdefghij0123456789 e 1:t 2:aa 1:y 1:r e

#### 6.3. GET\_FRIENDS

"Get friends" message allows retrieve friends of other NSW node. Queried node returns list of K nodes ordered by similarity with according similarity values. Node include option "me" in response if it is the closest. This is the important for "add\_friend" message. First node in list is nearest to term vector specified in request. Nodes are returned in a key "friends" as a list of strings. Each string containing "compact" format node information for a single node included port, hash\_id, similarity value.

There are no privacy concerns in this information because friend list except rarely cases does not include NSW nodes of the same peer. It follows that information what current peer is downloading is complicated for finding out.

The "term vector" is a list in bencode. Each element is info string where written in address, node id and similarity. As word's frequency is a float from range [0,1], it makes sense to convert values as string by copying bits in char array.

Key "**q\_id**" - id of querying NSW node

Key "**r\_id**" - id of recipient/response NSW node

Key "**term\_vector**" - term vector of querying node.

Key "**friends**" - K friends from routing table and their distance values.

Key "**token**" - The token value is a required argument for a future add\_friend request

Distance should be encoded as words frequency for term vector key.

**arguments**: {"q\_id" : "<querying nodes id>", "r\_id" : "<queried nodes id>", "term\_vector" : {"word1" : "freq1", "word2" : "freq2", "word3" : "freq3"}}

**response**: {"r\_id" : "<querying nodes id>", "q\_id" : "<queried nodes id>", "token":"aoeusnth", "friends" :

["<node 1 info string>", "<node 2 info string>", ..."<node K info string>"]}

Example Packets:

**get\_friends Query** = {"t":"aa", "y":"q", "q":"get\_friends", "a": {"q\_id":"abcdefghij0123456789", "r\_id":"0123456789abcdefghij", "term\_vector": {"word1":"q!.zw","word2":"q!.zw","word3":"q!.zw"}}

bencoded = d 1:a d 4:r\_id 20:0123456789abcdefghij 4:q\_id 20:abcdefghij0123456789 11:term\_vector d 5:word1 5:q!.zw 5:word2 5:q!.zw 5:word3 5:q!.zw e e 1:q 11:get\_friends 1:t 2:aa 1:y 1:q e

**get\_friends Response** = {"t":"aa", "y":"r", "r": {"r\_id":"abcdefghij0123456789", "q\_id":"0123456789abcdefghij", "token":"aoeusnth", "friends": ["abc123...", "def456...",...]}}

bencoded = d 1:r d 4:r\_id 20:abcdefghij0123456789 4:q\_id 20:0123456789abcdefghij 7:friends l 26:axje.u... 26:idhtnme...e e 1:t 2:aa 1:y 1:r e

#### 6.4. ADD\_FRIEND

Ask a node to add querying node into it friend list and provide own text description

It has the following parameters:

Key "**q\_id**" - id of querying node

Key "**r\_id**" - id of recipient/response node

Key "**description**" - term vector of querying node.

The node-recipient should check that it has already sent itself to same IP address and node id as answer on get\_friends request. After that node compares term vector with own one to verify sender is really in the range of K nearest. If for some reason sender is not closest errors should be returned. Else node adds information about sender in routing table and sends response with own term vector. As word's frequency is a float from range [0,1], it makes sense to convert values as string by copying bits in char array.

**arguments**: {"q\_id" : "<querying nodes id>", "r\_id" : "<queried nodes id>", "description" : {"word1" : "freq1", "word2" : "freq2", "word3" : "freq3"}}

**response**: {"q\_id" : "<queried nodes id>", "r\_id" : "<querying nodes id>", "description" : {"word1" : "freq5", "word2" : "freq6", "word3" : "freq7"}}

Example Packets:

**add\_friend Query** = {"t":"aa", "y":"q", "q":"add\_friend", "a": {"q\_id":"abcdefghij0123456789", "r\_id":"0123456789abcdefghij", "description": {"word1":"q!.zw","word2":"q!.zw","word3":"q!.zw"}}}

bencoded = d 1:a d 4:r\_id 20:0123456789abcdefghij 4:q\_id 20:abcdefghij0123456789 11:description <br />

d 5:word1 5:q!.zw 5:word2 5:q!.zw 5:word3 5:q!.zw e e 1:q 10:add\_friend 1:t 2:aa 1:y 1:q e

**add\_friend Response** = {"t":"aa", "y":"r", "r": {"q\_id":"0123456789abcdefghij", "r\_id":"abcdefghij0123456789", "description": {"word4":"q!.zw","word5":"q!.zw","word3":"q!.zw"}}}

bencoded = d 1:r d 4:q\_id 20:0123456789abcdefghij 4:r\_id 20:abcdefghij0123456789 11:description <br />

d 5:word4 5:q!.zw 5:word5 5:q!.zw 5:word3 5:q!.zw e e 1:t 2:aa 1:y 1:r e

#### 6.5. SEARCH\_QUERY

Ask a node just to provide description without adding into friend list has the following parameters:

Key "**q\_id**" - id of querying node

Key "**r\_id**" - id of recipient/response node

Key "**description**" - text of recipient and term vector of querying node

The node-recipient should check that it has already sent itself to same IP address and node id as answer on get\_friends request. After that it sends pure description as string.

**arguments**: {"q\_id" : "<querying nodes id>", "r\_id" : "<queried nodes id>", "description" : {"word1" : "freq1", "word2" : "freq2", "word3" : "freq3"}}

**response**: {"q\_id" : "<queried nodes id>", "r\_id" : "<querying nodes id>", "description":"text description of NSW node"}

Example Packets:

**search\_query Query** = {"t":"aa", "y":"q", "q":"add\_friend", "a": {"q\_id":"abcdefghij0123456789", "r\_id":"0123456789abcdefghij", "description": {"word1":"q!.zw","word2":"q!.zw","word3":"q!.zw"}}}

bencoded = d 1:a d 4:r\_id 20:0123456789abcdefghij 4:q\_id 20:abcdefghij0123456789 11:description <br />

d 5:word1 5:q!.zw 5:word2 5:q!.zw 5:word3 5:q!.zw e e 1:q 10:add\_friend 1:t 2:aa 1:y 1:q e

**search\_query Response** = {"t":"aa", "y":"r", "r": {"q\_id":"0123456789abcdefghij", "r\_id":"abcdefghij0123456789", "description":"text description of NSW node"}}

bencoded = d 1:r d 4:q\_id 20:0123456789abcdefghij 4:r\_id 20:abcdefghij0123456789 11:description 28:text description of NSW node <br />

e 1:t 2:aa 1:y 1:r e

### 7. Security

===========

TBD

Security is weak element of NSW extention