Time-limited data storing in peer to peer networks

Scherbakov Konstantin
SPbSU
Saint Petersburg,
Russia
konstantin.scherbakov@gmail.com

ABSTRACT

Different techniques of using peer to peer networks are mentioned. The process of data distribution in peer to peer network by the single node's initiative for purpose of further downloading by this node's owner is described. A common model of special peer to peer network (UDD network), developed for single node initiated data distribution, with its main characteristics and used algorithms is introduced. Some ways of improving existing peer to peer networks for the purpose of compliance to UDD network specification are mentioned. Experimental results of single node's initiated data distribution in one of the existing peer to peer networks (with our improvements applied) are analyzed.

Introduction and related work

Pure and hybrid peer to peer (p2p) networks are widely used now in our life. There are at least two main ways to use them:

- for grid/cloud computing/storage (Amazon S3, Nirvanix SDN, Gridnut, Wuala etc.) [1,5,9,13,19,25]
- for data exchange between interconnected nodes (bittorrent, exeem, ed2k, kademlia, gnutella 1,2 etc) [2,4,6,7,21,22]

If p2p network is designed for data exchange, new nodes usually connect to it for the purpose of retrieving some files, they are interested in, or share to other nodes some files, they want to. But there is at least one more alternative way of using such type of p2p networks. One node can connect to network for storing its own data/files over other nodes for the purpose of retrieving this data by node's owner from any other place. This data may be uninteresting for other nodes of network and may be encrypted. So let's call such type of data the *u-data*. This way is rarely used because of lack of data access control, data availability control and u-data fast distribution mechanisms in existing p2p networks.

Our main goal is to introduce the architecture of prototype of such p2p network and search request routing/data storing/indexing protocols for it, that allows any node to store securely its u-data over the network for some fixed time interval and also grants this node's owner ability to retrieve his data from any other location during fixed period of time mentioned below and control availability of this data over the network during its limited storing period. Using of these mechanisms by particular node should depend of some coefficient of node's usefulness for the whole network. So nodes, that are more useful for the network, should have opportunity to use these mechanisms more often and more completely.

UDD P2P network model

Let's assume that one person or group of persons have some u-data, that is very useful for this person or group of persons, but not interesting for other participants of our future u-data distribution (UDD) p2p network. This person wants to have access to this data from any place, where he can connect to internet (and to our UDD p2p network too). He also wants to have ability to extend his data storing time and he expects that there is some mechanism used in this network that prevents with high probability his data from being deleted from all network nodes during its limited storing time.

What properties we expect for our UDD p2p network? Let's assume that our network will have the properties listed below:

1. Network nodes should have their own limited long-term memory
2. Network nodes should have access to some computing resources (for example their own CPU) to calculate data content and keyword hashes
3. Network should have some mechanisms for u-data storing and effective distributing and deleting
4. Network should provide high probability data disappearance preventing mechanism
5. Network should have mechanisms of effective data search request routing and metadata storing [3,8,10-12,14,15,17,20,23,24,26,27]
6. Network nodes may connect and disconnect from network constantly and therefore network provide some mechanisms of retrieval node connect/disconnect rate

This is ideal network for our purposes. Real p2p networks (especially file sharing) often have 3-4 of these properties, but we won't examine them at this point. It's only important to note, that some of this networks may be rather easily extended in special way to have all properties of our ideal p2p u-data storing network [16]. It's important to introduce and describe the method of u-data distribution in UDD network initiated by some specific node.

In usual p2p network data distribution process usually

have some general stages:

- Data hashing (data content hashing, data description keyword extraction and hashing etc.)
- Data announcing (more precisely this stage should be called "data hashes announcing")
- Waiting for external requests for announced data from other network nodes
- Data uploading to external network nodes interesting in data announced

It's a good way of distribution process for data that may be interesting and useful for other peers in usual file sharing p2p network, where waiting for external requests stage hasn't infinite durability for the data being distributed. But we assumed that our data is uninteresting for other nodes. Furthermore, our data may be encrypted by its owner to prevent effective using of it by other network node's owners. Therefore 3rd stage of usual data distribution process will have infinite durability for u-data in usual file sharing network. So how we can effectively distribute data in our ideal network? Nodes have some limited long-term memory. If we try to send our data in all accessible nodes, some of them may reject it, because of lack of free disc space. Our first goal is to share u-data between N nodes (including the initial one), where N is defined by data and initial node's (n_init) owner (N should be limited according to the node n_init usefulness coefficient, mentioned above). Lets assume that n_init have M neighbors. If M>N we will ask them about their free space and ask them to reserve *sizeof(u-data)* bytes for our u-data storing with M queries and get M_ok results with satisfying answer. If M_ok < N, we will ask this nodes for their neighbor lists and then repeat our first step. If M<N, we can ask them for neighbors first and then ask extended node list for free space reservation. But we can save some network bandwidth by uniting these two types of requests into a single one. So n_init can ask its neighbors to reserve some free space for its u-data and in the same request it can ask them to forward this request to their neighbors. After receiving at least M satisfying results, n_init sends to each of them full list of nodes from this list and offers to receive his data. M nodes start to receive this data, using swarm cooperation. After that they receives a unique id's, which are actually a structure with some fields: data hash, data lifetime/finish storing date, replication coordinator priority/id (this value will help to determine current replication coordinator for stored u-data), M, owner node's data access key (a special access key for this portion of data only), owner node's id key, search keywords, etc. This process is briefly described in the code listing below

Listing 1:

```
$neighbors_arr = get_neighbors();
//neighbor list

$m_ok=0;

foreach ($neighbors_arr as
$neighbor_num=>$neighbor_value)

{

  if (reservespacerequest(

  &$neighbors_arr[$neighbor_num],&$m_ok,

  $space, $datahash)==1)

  $neighbors_arr[$neighbor_num]

  ['reserved']=1;

  $neighbors_arr[$neighbor_num]

  ['checked']=1;

}

$cur_neighbor=
get_firstkey(&$neighbors_arr);

while ($m_ok<$n)

{

  addneighbors_callback(&$neighbors_arr,

  &$m_ok,$cur_neighbor,

  'reservespacerequest');

  if(($cur_neighbor=get_nextkey(

  &$neighbors_arr,$cur_neighbor))===

  false) break;

}

$invited=send_download_invitation(
&$neighbors_arr);

/*some code deleted from listing*/

$resultnodes=send_data_info(
&$neighbors_arr);
```

As was mentioned above our ideal network can be constructed by extending some real p2p file sharing network (for example bittorrent of ed2k/kademlia network). But if we realize a possibility of distributing u-data in some client application for these networks without any limitation, very few nodes will use it, because while being unlimited, this process can make a lot of parasitic traffic in the network and slow down downloading and uploading of usual files. So if we want to extend these networks with u-data distribution ability, we need to describe some methods that can guarantee limited use of this function, for example by useful for whole network nodes only. Usually coefficient of usefulness is represented by formula

coef_usf=node_upl/node_dwn,

where node_down>0, and infinity otherwise.

When any nodes download some data they need, their node_dwn values increases, and when they upload some data to other nodes, their node_upl values increases. These values can never decrease. For our extended network, we can change this formula to

*fair_coef_usf = (u_usual+ deltads*d_special) /*

*(d_usual + deltaus*u_special),*

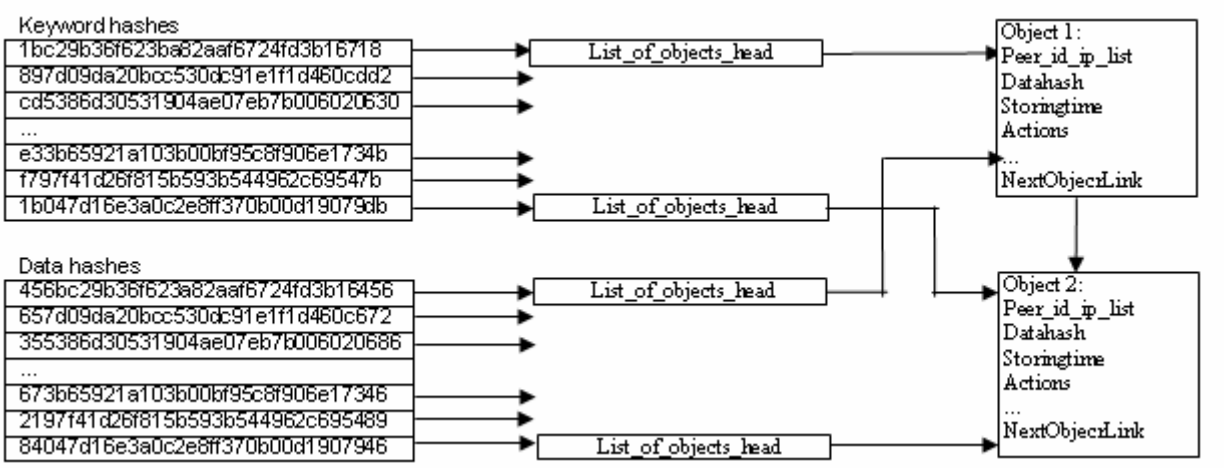
where u_usual – outgoing traffic value for usual data, d_usual – incoming traffic value for usual data, u_special and d_special – incoming and outcoming traffic value for special data (distributed over our u-data distribution mechanism), deltads, deltaus – special weight coefficients, introduced for correcting value of fair_coef_usf after using by some node possibility to distribute u-data .

Now let's return to the end of u-data distribution process. Let's assume that our network have some special nodes, that allow storing data indices (it's true, if we'll extent bittorrent or ed2k network, that are now).

All M nodes with our data stored on it sends data hash, finish storing date, search keywords to their neighbor nodes, which are marked as indexing nodes, which stores these values in a special data structure, you can see on the scheme 1 below.

As we can see on this scheme, our special node has 2 hash tables. First hash table contains hashes of search keywords, stored in dynamic array as array keys. Each value in this array contains link to a special list of pointers to objects, where each object contains content hash for data, relevant to this keyword, data note and filename, node id's or addresses, where this data is stored. We can store ids for nodes with dynamic ip addresses and raw ip addresses for nodes with the static ones. But if we only have hash table for keywords, our special nodes will be useless for search by data content hash request routing. So we'll create a second hash table for this purpose. It contains known data content hashes links to special node, with links to the head of special object list, which also contains links to real objects with data about data hash, keyword hash, node id's and/or ips, mentioned above. So, when search request arrives, we split it to keywords and search for their hashes in first table. Then we get data filenames, keyword relevance in some way, data notes and send it back to node, that initiated this request and let it decide, what to do with the result retrieved. When hash search request arrives, we send back to the request initiator ids / ip addresses of nodes that can store this data.

How we can use this structure for updating storing-time of u-data distributed in our network? Node n_init should send special request with data hash and new storing time to all known nodes with our hash table structures. Then these special nodes should find a record about this data by its content hash. And finally they should update storing time for this record and send update request to known nodes with this data stored. This is non-guarantied way to update data storing time, but our goal is to update data on some nodes. All other work should do current data availability coordinator.



Listing 2:

```
$reqtype=get_rec_type($request);
if ($rectype=='updatetime' &&
($newtime>time()+$delay))
{
  $dataobj=0;
  search_by_datahash($hashstructure2,
  $hash,&$dataobj);
  update_storingtime_delayed(&$dataonj,
  $newtime,$delay);
  mark_for_update(&$dataonj);
  send_updatetimerec_delayed(&$dataonj,
  $newtime);
}
elseif ($rectype=='deleterec')
{
  $dataobj=0;
  search_by_datahash(&$hashstructure2,
  $hash,&$dataobj);
  send_deleterec_delayed(&$dataonj,
  $delay);
  delete_delayed(&$dataonj,$delay);
}
```

```
/* $delay represents time in seconds,
after which this object is actually
deleted after it was disabled for search
requests */
```

In the listing 2 we can see code that allows indexing nodes to update or delete objects with data hashes and nodes ids/ips lists. It's significant that objects are not deleted immediately, so we only mark them as deleted and set some delay time, after that it will be actually deleted during the regular indexing node's maintenance process. This process should take place regularly in the periods of low CPU/network/etc load of indexing node. While maintenance, indexing node should check all objects for it's data storing time and if necessary, physically delete them, if undelete flag is not set. Else if this flag is set, node indexing should take away delete flag and update storing time, if it's higher than current time, and if there is no delete flag for this or higher storing time for this data hash, else node should also delete object permanently.

So, how we can use this structure for searching u-data, distributed over the network and for updating its storing time / non-guaranteed deleting etc.?

Here you can see this process:

Listing 3:

```
if (is_server_load_low())
{
  foreach ($hashstructure2 as $hkey =>
  $hvalue)
  {
    $actionlist=getactions(
    &$hashstructure2[$hkey]);
    if ($actionlist)
      writelog(executeactions(
      &$hashstructure2[$hkey],
      $actionlist));
    while (!is_server_load_low())
    {
      sleep(5);
    }
  }
}
```

We have mentioned data coordinator below. Let's briefly describe its functions. Regularly all nodes, storing our u-data sends requests to current coordinator to check its availability, tell it that they are available and receive a list of other nodes with u-data stored. Coordinator monitors these requests and has a list of currently active nodes with u-data presented. If coordinator suddenly disconnects, other nodes elects a new one by special data id, they have. They use last received nodes list in this process. Node with minimum data id (this may be UNIX time for example) wins the election and became a new coordinator. When the old coordinator arrives back in network, he sends requests to known nodes with u-data stored and receives new coordinator address. Then he tells new coordinator to tell other nodes about new old coordinator send current node list and became a regular node.

Experimental results

And now let's make some experiments with our data distribution mechanism in the real network. We have extended existing bittorrent client bittornado, torrent tracker tbdev [2,21,22] and a special tool to emulate large number of different nodes (this tool is written on PHP). We are using 2 servers: C2D E4400, 3GB ram, Centos 5.0 and C2D E6400, 3GB ram, FC6. We know some statistics for the normal bittorrent network, based on this 2 servers: network average size = 4100 peers; about 1700 peers connected to network with intervals more 24 hrs. Average peer renewal speed: 0.08 peer per second. Average incoming/outcoming speed of all peers = 1.4 mbit/s

Nobody other then us was given modified p2p network client, so we will create virtual nodes on server 1, make server 2 indexing node and start to distribute data over our virtual network with most characteristics equal to the real one.

Firstly we'll assume, that all peers have appropriate disc space for our data and we wont's actually save it on it's discs. We'll only save data hashes, storing time and indexes instead of it. 1700/4100 * 100% roughly = 41%. So in all our experiments 41% of peers will be always connected to network.

In first two sets of our experiments we'll set and fix the coordinators updating interval to 1 hour and start to distribute data to varying number of nodes. We'll also vary average number of connected nodes (in experiment set two). Our goal is to determine conditions, when last node with our data will leave network and it became inaccessible and how many nodes we need to have 25% of nodes at the end of one coordinator update period.

Then we'll also vary coordinator updating interval and will determine the same conditions, when all stored data will disappear. But firstly we'll assume that our network doesn't have a constantly connected (core) nodes.

Every experiment we'll repeat 100 times to determine best and worst results for current conditions.

In the first set of experiments (100 nodes connected to network in average, coordinator update time = 1 hour; every hour 59 random nodes leaves network and 59 other connects)[1] we can see, that when N (total nodes to

[1] Raw data for this set of experiments can be found at http://195.70.211.9/syrcodis09_set1.txt

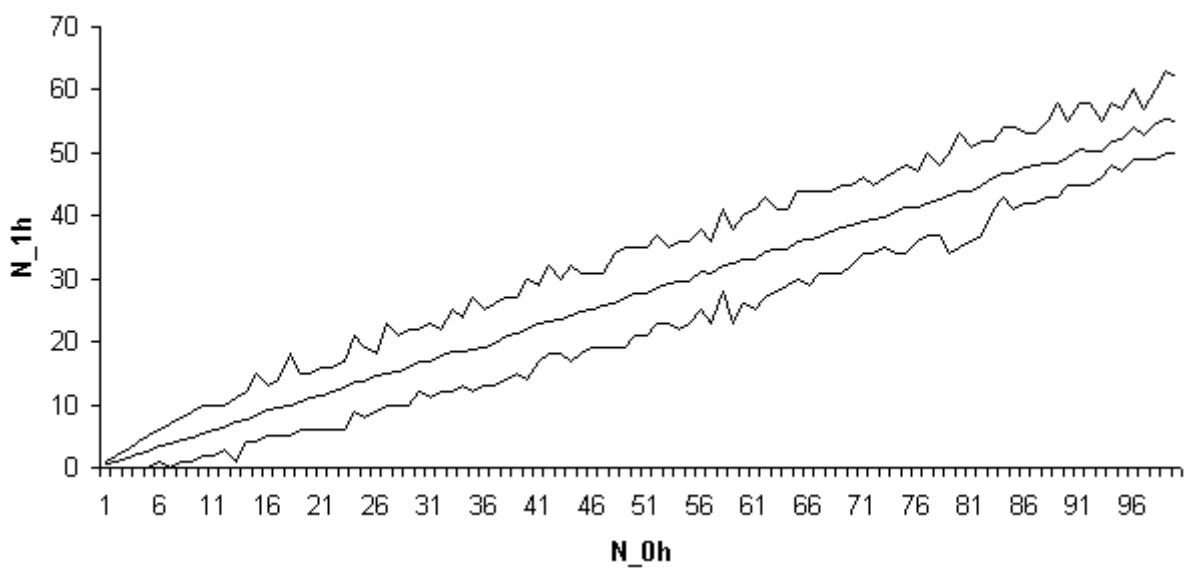


Diagram 1. First set of experiments.

which our data is distributed initially) reaches value of 14, more than 25% of nodes will still have our data on it after 1 hour in the worst result acquired. Theoretically N should reach the value of 60 to give as a guarantee of data saving after one coordinator update period at least on one node, but practical results are better, because probability of the event "all nodes with data leave our network after one coordinator update period" decreases exponentially (and will be about O(10^(-59*2)) for N=59). So in real network we don't need to distribute data on such huge amount of nodes to save our data with probability very close to 1.

see slightly higher number of minimum required nodes for data saving on 25% of nodes at the end of coordinator update period. It's close to 25 nodes, because in this set of experiments 59 unique nodes leaves the network, and in set 1 this value is distributed in the interval (1,59).

Now let's vary total number of connected nodes with other properties of network equal to set 2. At the Diagram 3, which represents the 3rd set of experiments[3] we can see results for 4000 connected nodes and for N from 1 to 2996 with step 85. While N reaches value of 86, we have more than 50% of nodes with our data alive

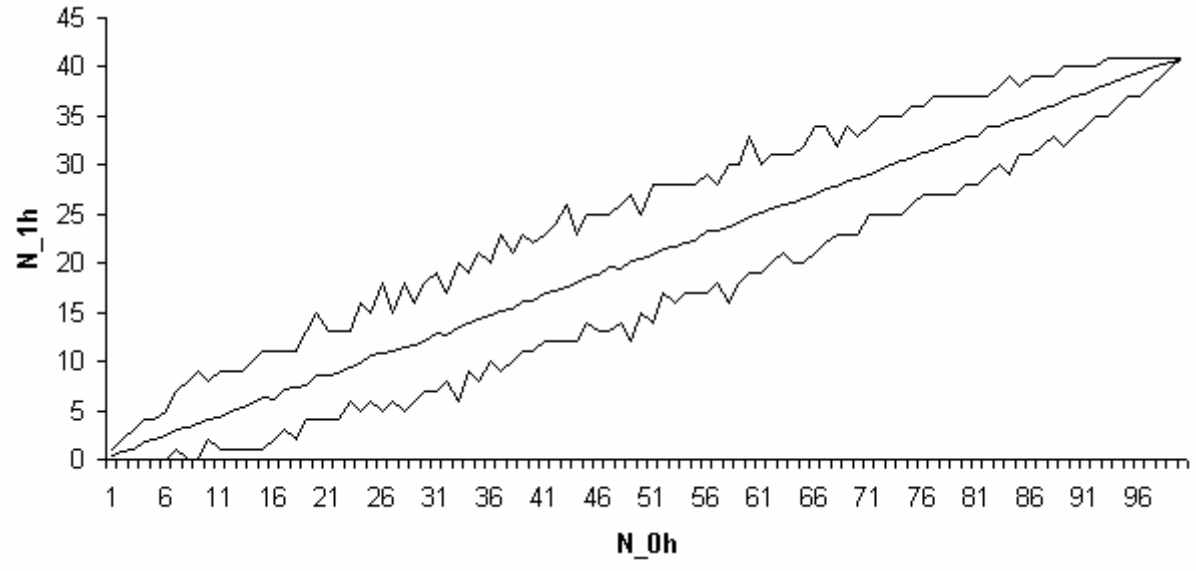


Diagram 2. Second set of experiments.

When taking a look to diagram 2 (second set of experiments, where we have 41 persistently connected nodes, and other properties are equal to set 1)[2] we can

after coordinator update period

While network connect/disconnect rate is fixed, it's

[2] Raw data for this set of experiments can be found at http://195.70.211.9/syrcodis09_set2.txt

[3] Raw data for this set of experiments can be found at http://195.70.211.9/syrcodis09_set3.txt

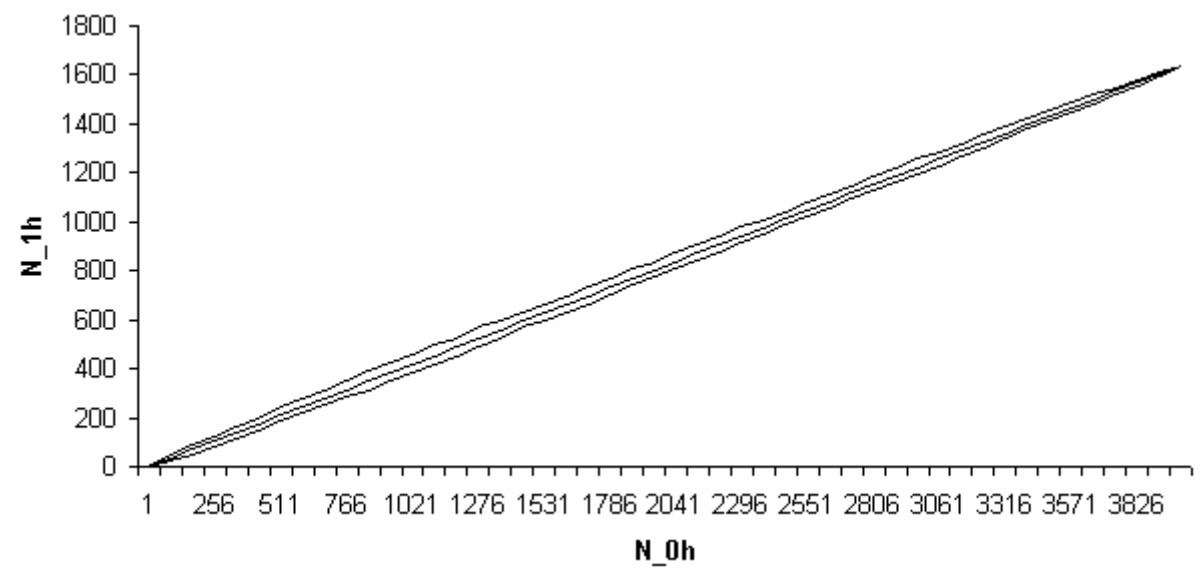


Diagram 3. Third set of experiments.

better to have more non-persistently connected nodes for minimizing the number of nodes, to which our data should be distributed for having the probability of saving very close to 1 after one coordinator update period. This probability will decrease with time very slowly, so we'll have its high enough at the end of our data storing interval, because it's not equal to infinity. More experiments are required to determine maximum satisfying storing time interval for high probability of data saving. We can also say that varying coordinator update interval can help us to increase the probability of data saving. It's important to rightly determine update interval before distributing any data in network and vary this interval while data life cycle to minimize the number of nodes to which data is distributed and save the network bandwidth and node's computing resources. We can do this by sending special requests to indexing nodes for example (we have one of them in our experiments – it's an extended bittorrent tracker). These actions will make a little additional non data-transfer traffic over the network, but will save us significantly more traffic between nodes.

Conclusions

So, we have described a model of ideal p2p network for data distribution initiated by a single node. We also introduced some methods, that should be used in such type of network and make three series of experiments with good results. Our next work will be concerned to examining more deeply routing mechanisms in that type of networks and introducing methods that will allow making data distribution and retrieval more secure.

REFERENCES

[1] Amazon S3 official documentation: http://docs.amazonwebservices.com/AmazonS3/2006-03-01/

[2] BitTorrent full specification (version 1.0). http://wiki.theory.org/BitTorrentSpecification

[3] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In ICDCS, July 2002.

[4] Exeem project specification: http://www.exeem.it.

[5] I. Foster. Peer to Peer & Grid Computing. Talk at Internet2 Peer to Peer Workshop, January 30, 2002

[6] Gnutella project specification: http://www.gnutella.com.

[7] Kademlia: A Design Specification. http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html

[8] V. Kalogeraki, D. Gunopulos, D. Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks. In CIKM, 2002.

[9] J. Kubiatowicz, D. Bindel, Y. Chen. Ocean-store: An architecture for global-scale persistent storage. In ASPLOS, 2000.

[10] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In ICS, 2002.

[11] D. Menasce and L. Kanchanapalli. Probabilistic Scalable P2P Resource Location Services. SIGMETRICS Perf. Eval. Review, 2002.

[12] I.Nekrestyanov. Distributed search in topic-oriented document collections. In SCI'99, volume 4, pages 377-383, Orlando, Florida, USA, August 1999.

[13] Nirvanix SDN official documentation: http://nirvanix.com/sdn.aspx

[14] S. Ratnasamy, P. Francis, M. Handley. A scalable content-addressable network. In ACM SIGCOMM, August 2001.

[15] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Middleware, 2001.

[16] K. Scherbakov. Search request routing in Bittorrent and other P2P based file sharing networks, SYRCoDIS 2008, Saint-Petersburg, Russia

[17] I. Stoica, R. Morris, D. Karger. Chord: A scalable peer-to-peer lookup service for internet applications. In Proc. ACM SIGCOMM, 2001.

[18] M. Stokes. Gnutella2 Specifications Part One. http://gnutella2.com/gnutella2_search.htm.

[19] D. Talia, P. Trunfio. A P2P Grid Services-Based Protocol: Design and Evaluation. Euro-Par 2004

[20] A. S. Tanenbaum. Computer Networks. Pren-tice Hall, 1996.

[21] TBSource official documentation: http://www.tb-source.info

[22] TorrentPier official documentation: http://torrentpier.info

[23] D. Tsoumakos and N. Roussopoulos. Adaptive Probabilistic Search for Peer-to-Peer Networks. In 3rd IEEE Int-l Conference on P2P Computing, 2003.

[24] D. Tsoumakos, N. Roussopoulos. Analysis and comparison of P2P search methods. Proceedings of the 1st international conference on Scalable information systems, Hong Kong, 2006

[25] Wuala project official documentation: http://www.wuala.com/en/about/

[26] B. Yang and H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. In ICDCS, 2002.

[27] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location androuting. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001