periodica polytechnica

Electrical Engineering
54/3-4 (2010) 87–92
doi: 10.3311/pp.ee.2010-3-4.01
web: http://www.pp.bme.hu/ee

RESEARCH ARTICLE

Enhancing the Kademlia P2P Network

Zoltán Czirkos / *Gábor* Hosszú

Received 2011-09-13

Abstract

Distributed Hash Tables (DHT's) are sophisticated Peer-to-Peer (P2P) overlay networks. Such overlays have the ability to retrieve stored data in a limited time, usually in a logarithmic number of steps. However in contrast to the well-known Gnutella and FastTrack networks, these can only locate data quickly, if the key associated with the data requested is accurately specified. In this article we analyze the reliability of the Kademlia network, and describe our model, which can be used to determine its system-wide configuration parameters. We also present a novel algorithm that implements broadcast messages in Kademlia. The developed algorithm ensures reliable delivery of broadcast messages in an error prone environment. Broadcast messaging is an elementary service in an overlay network. Using broadcast messages, queries of any key type or part of key, can be realized.

Keywords

P2P · DHT · Kademlia · broadcast

Zoltán Czirkos

Department of Electron Devices, BME, H-1117, Magyar tudósok körútja 2., Hungary

Gábor Hosszú

Department of Electron Devices, BME, H-1117, Magyar Tudósok körútja 2., Hungary

1 Introduction

Kademlia is a peer-to-peer distributed hash table with many desirable features [1]. It minimizes the number of system-wide configuration parameters, as well as the network maintenance overhead traffic. Routing information used by its nodes is spread as a side-effect of delivering payload traffic, i.e. data store and lookup requests. Due to its design, it is also partially resilient to Sybil attacks [11].

Kademlia does not require its nodes to pass their stored key-value pairs over to neighboring nodes when leaving the network. Therefore it gives a probabilistic guarantee for being able to retrieve data stored in it. To enhance data retention time, replication is used, which means that data is not stored only at the node determined by its key, but at a range of nodes closest to the key. The level of replication is a small integer value designated by k. The value of k is the only system-wide configuration parameter required. It is estimated by the designers of Kademlia using average peer availability and session duration data collected from the Gnutella network [1].

In this article we show that replication can also be used to deal with network errors. We describe a method named KPM (Kademlia Performance Model), which can be used to calculate the value of k for any given reliability demand and a given number of average network errors. We present a simulation to validate our results and give an insight about how replication makes it possible in Kademlia to enhance data availability and retention time.

In a further addition to the possibilities of Kademlia, we designed broadcast algorithms on top of its binary tree topology. Our broadcast algorithms give probabilistic guarantees to deliver the broadcast message to all nodes in the network. We compare our algorithms in terms of speed, efficiency and network load induced.

The rest of the paper is organized as follows. First we present an overview of the Kademlia network. Then in the next section we explain our Kademlia Performance Model. The broadcast algorithms, which can be implemented on top of the binary tree topology, are described afterwards.

2 Overview of the Kademlia Overlay Network

Peer-to-peer networks are decentralized, virtual overlay networks, which enable applications to communicate among each other using a network topology that is more feasible to them than the one of the underlying physical network.

Applications running the P2P software are usually called peers or nodes. P2P networks can be either structured or unstructured. In *unstructured networks*, nodes can easily be dispensed; the overlay handles parting and failing of nodes flexibly. Queries for data items are handled by all nodes in such networks, by means of mechanisms built into the substrate [2]. Gnutella, Freenet and FastTrack are typical examples for unstructured overlays [12].

Structured substrates are Distributed Hash Tables (DHT's). The connection between nodes, i.e. the topology of the network is precisely defined. These overlays store key-value pairs, and allow quick lookup of any value associated with a given key. Each key-value pair is stored at a node, given by an appropriate algorithm. Each node is assigned a node identifier (NodeID) from a large range of integer numbers. Similarly, pieces of information are also assigned a key, which can be a hashed value of a file name. This key can be called a file identifier (FileID), which has the same number of bits as the NodeID. Every node stores those key-value pairs having their hashed keys closest to its NodeID, i.e. their FileID's closest to the NodeID. Given a precise key, the exact location of a file can be determined.

Structured networks differ from each other in their methods of overlay management, routing algorithm and distance function applied for measuring the relation of participants. Nodes of a Kademlia DHT can be represented with a *binary tree* [3]. In their routing tables they maintain a list of possible connections (IP address and port number) for every subtree, called *k-buckets*. The size of these lists is a small integer k, which is the only system-wide configuration parameter. Large networks can have subtrees with a size much larger than k. A single node has relatively smaller knowledge for large subtrees, and it knows its close neighbors perfectly.

Nodes in the Kademlia overlay have comparatively greater knowledge about their closer neighbors. Routing is implemented as follows. To find a node with a specific NodeID, one successively queries nodes closer and closer to the destination, by each query addressing a smaller subtree towards it. Consider Fig. 1 for an example. If the black node with address 00110 wishes to send a message to node 11100, it has to send a query to any node with address 1****, which has more precise knowledge about nodes with addresses 11***. The node queried replies with a k-bucket, containing IP addresses of nodes which have their NodeIDs closer to the destination. Then the sender chooses node 11000, which is also queried and so on. The destination of the message is looked up in O(log n) steps.

The distance (which is measured on the topology, and is not to be confused with the geographical locations of nodes) between two identifiers is calculated with the exclusive-or (XOR) function. *The magnitude of the distance is the height of the smallest subtree, which contains both nodes.* If two nodes are A and B, the distance between them is denoted by $d(A, B)$that is calculated using the XOR metric, i.e. with the formula $d(A, B) = A\text{XOR}B$.

3 Properties of Routing in Kademlia

The routing of Kademlia is more flexible than the methods used in other DHT systems. The nodes of most types of DHT networks communicate only with their neighbors, i.e. when a message is sent to some distant node; it is usually forwarded from one node to another until it arrives at its final destination. However this is not the case for the Kademlia network. In Kademlia, the routing procedure presented above works rather like an IP address lookup procedure [1]. If a node is willing to communicate with another one, it will ask other nodes successively closer to the destination for the IP address and port number of the node which is the destination of its message. Then the two nodes communicate directly by using UDP packets. This makes overlay management in Kademlia totally different from methods used in other networks.

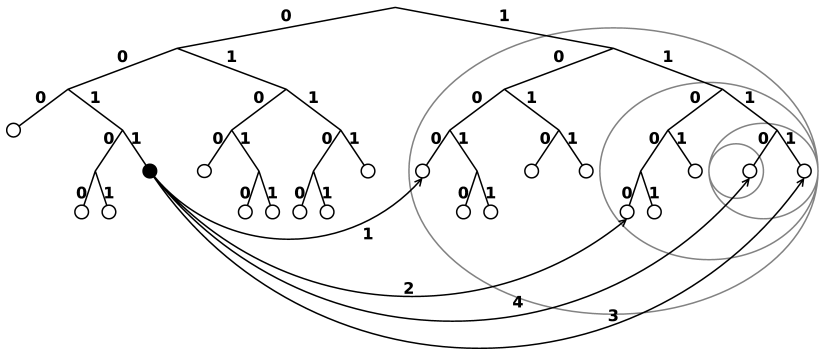
Kademlia nodes leaving the overlay do not pass their key-value pairs over to other neighbors. If a node quits the network, data it stored would also disappear, unless it is stored at multiple locations.

The method of routing, namely nodes looking up IP addresses of each other and communicating directly, simplifies management of data replication. It can be implemented simply by the publisher node requesting not only the closest node to the key, but a range of nodes to store the key-value pair. The size of this range is also k, equals to the size of k-buckets. The choice of k therefore affects also the stability of the network, as well as the availability of the key-value pairs stored. Given the fact that in a DHT the availability of a specific node directly implies the availability of a key, *it is feasible that the level of replication is the same as the degree of stability*, namely k, the size of the k-buckets [6].

The cost of increasing replication to achieve higher reliability of the overlay increases the network traffic. In the Kademlia overlay, nodes are responsible to store their data to be shared at different destinations. The cost of a single store message is the cost of the message itself plus the cost of the required lookups to find the destination node. When the level of replication increased, the per-store cost of messages increases *slower than linearly* (i.e. it is less than O(x)), as the k-buckets resulting from lookups required for different destinations can have items in common. This is caused by the iterative lookup procedure Kademlia uses [1] – the route on which a node lookup is carried out may share subtrees with another lookup.

Lookups in Kademlia have the advantage of keeping the routing tables of nodes fresh [1]. In a low traffic network, the routing tables also benefit from using replication.

Fig. 1. Locating a node in the Kademlia overlay.



3.1 A Probabilistic Model of Network Errors

Replication in Kademlia is more important than in other overlay networks. Some nodes might be unable to connect each other due to packet losses, packet filtering, network address translation or other reasons [3], [4]. It is possible that the IP address of a destination node is looked up, but after that the sender is not able to communicate with it. Replication can solve this problem, as *if data is not only stored at one node rather at k different nodes, the probability for the sets of nodes for data store and the sets of nodes of data lookup requests to have a common one in their intersection is much higher.* Replication also increases data availability if the routing tables are partially incorrect, which is the case in high churn networks [5], where nodes join and quit the network frequently. This is caused by the side-effect of network traffic keeping the routing table elements fresh in Kademlia [1].

To model how replication solves problems of availability and network packet losses, we take the following facts into consideration. Nodes joining the Kademlia network choose a specific application level identifier for themselves randomly. Also the output of hash functions is a uniformly distributed, seemingly random number. This means that the data to be stored seems to end up at randomly selected nodes.

As the properties of hash functions, and the random selection of NodeID's result in STORE requests ending up at random locations, the exact association of error ratios to nodes – i.e. which specific pairs of nodes are able to communicate and which pairs not – is indifferent. Only the global distribution of the errors is important, since some nodes can receive most of the messages, some not. For Kademlia, the exact distribution of network errors only affects numeric results, but not the fact that replication can make stored data available with higher possibility. Various distributions yield different availability ratios for each node in the system. If part of the nodes are fully accessible (for example, they have public IP addresses [3]), one of them will be among the ones designated to store the key currently in question with high probability, provided that the level of replication is sufficiently high.

If the Kademlia overlay implements replication, a node has more than one, exactly k opportunities to store or retrieve data. Practically speaking, the probability of correct lookups denoted with P increases. Calculating the probability of all lookups failing (P' is the probability of a single lookup being successful) to estimate the probability of at least one correct lookup, we get Eq. (1).

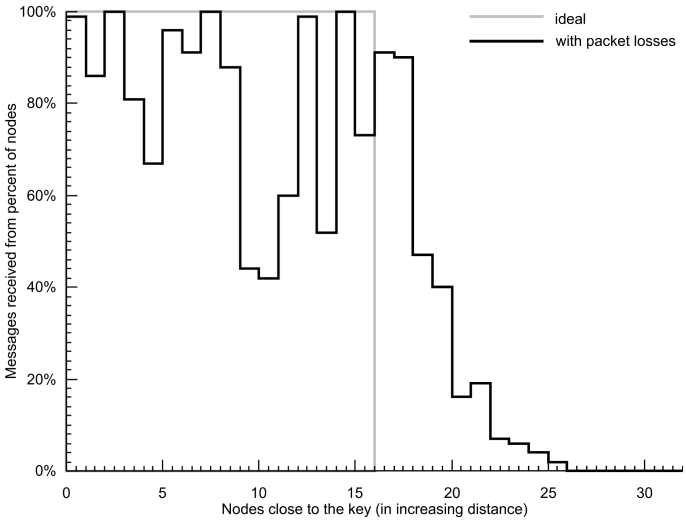


Fig. 2. Storing keys in a simulated Kademlia overlay.

$$P = 1 - \left(1 - P'\right)^k \tag{1}$$

This gives us the probability of successful lookup despite network errors. In this formula, k is the level of replication, the number of nodes storing a given key-value pair.

By solving Eq.(1) for k, the necessary replication factor can be estimated, if the ratio and distribution of network errors (P') and required probability of correctness (P) are given. The replication factor k, for which we gave above a method to estimate, is essentially the same as the size of the k-buckets in Kademlia. The model we presented here named KPM (Kademlia Performance Model) can be used to determine this configuration parameter k for such an overlay, as it is a trade-off between dependability and induced network traffic.

3.2 Verification of the Kademlia Performance Model by Simulation

To verify the model presented above, we developed an application specific simulator, KadSim. The program simulates the following scenario: in a Kademlia overlay, every node decides to send a STORE request to a selected node (to k nodes closest to the selected one, when using replication). By visualizing the number of packets received by nodes closest to the key (of

which can be more than one, if one of the senders could not reach the exact destination of the message), we get an overview of network errors. In the overlay simulated by KadSim, every node looks up the node closest to this key, not counting nodes that are unreachable.

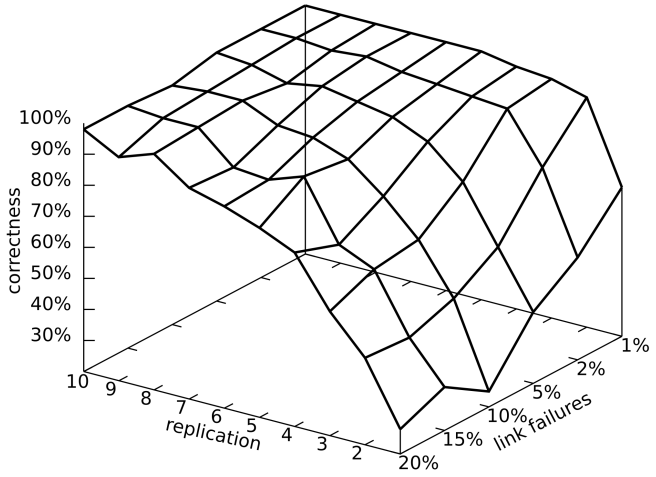


Fig. 3. Correctness of Kademlia lookups.

At the end of the simulation, KadSim sorts nodes by their distance to the hashed key, and plots the number of messages received by each. Ideally, if all network connections work, the plot shows a step function: the k closest nodes receive n messages, and others receive zero, where n is the number of all nodes in the overlay. As an example to demonstrate what happens when network errors hinder the data store and lookup procedures, Fig. 2 shows the results for a simulation with 20% of the links failing, with replication $k = 16$. On Fig. 2, if one of the nodes is unable to send the message to the 12^{th} and 15^{th} closest nodes, it will send it to the 16^{th} and 17^{th}. Those are out of the 0..15 range of the $k = 16$ replication, but some nodes cannot reach all neighbors in the 0..15 range, and therefore they decide to send their store requests to others.

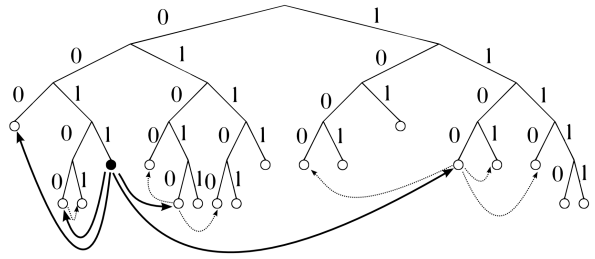


Fig. 4. The broadcast algorithm implemented for the Kademlia overlay.

Simulation results show that a relatively low, $k = 8$ replication already ensures that there will be a node, which is able to receive all messages (Fig. 3). This might look too much for an overlay counting a hundred of participants. Nevertheless, increasing the number of nodes does not involve the need to increase the level of replication, since the level required is only determined by the ratio of failing links, and not the number of participant nodes. With high probability there will be a node that is reachable by every other participant.

4 Broadcast in P2P Overlays

Implementing broadcast (one to all) messages in P2P networks is rare, due to the large number of nodes. Still, there are applications, which require this type of communication. It is important to note that the term 'broadcast' can have two different meanings in the context of P2P networks. Broadcast messages are one to all messages, the type of communication when a single node sends some piece of data (maybe with the help of other nodes) to every other entity in the network. When referring to multimedia content delivery, 'broadcasting' content means delivering media streams to some, usually most but not all, nodes in the network. The latter is called broadcast for historical reasons (as we use the same term for radio and television transmissions [9]), but is essentially a multicast-type communication. In this article, we use the term 'broadcast' in its 'one to all' meaning.

The inherent topology of structured networks is a useful substrate to implement an efficient broadcast service. As messages can be duplicated at any node, the broadcast will take place in logarithmically many steps, and logarithmic time. A further advantage of the Kademlia overlay is that there is no need to initiate new connections during a broadcast. The list of ecently seen nodes can be used. The topology of the overlay is essentially an implicit multicast tree.

In the following sections three different algorithms to implement broadcast service in Kademlia will be presented. In our tests and simulations we treat packet losses as terminal, i.e. nodes do not try to detect and resend lost messages. This enables us to study how the algorithms perform in a short timeframe. Detecting a packet loss usually takes many seconds; that is enough for a complete broadcast sequence to take place, as the number of nodes receiving the message grows exponentially in time.

First algorithm: broadcast using flooding. All nodes send received messages to any other nodes they know. As a single message can be received in duplicates, every broadcast request is tagged with a unique identifier. Already known messages are dropped by nodes. This solution is simple, but it generates a lot of network traffic, especially when k-buckets are large. It has no practical use, but is rather a theoretical reference; by simulating this method on an overlay, the time the broadcast requires can be seen.

Second algorithm: broadcast using the topology. Every subtree in the Kademlia overlay is assigned a node, which is responsible for broadcasting the message in its own tree (Fig. 4). In Fig. 4 the node with the identifier (00110, black dot) initiates the broadcast by sending it to one freely chosen node from each of its k-buckets. The nodes in the example are 11000, 01010, 00100 and 00000. The nodes receiving the message are responsible for sending them on in their own subtrees, which are

1****, 01***, 000** and 0010*. This is shown using dashed lines. Broadcast using this method will be finished in logarithmic time.

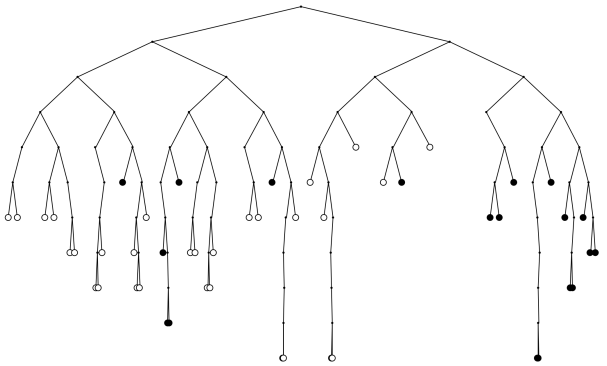


Fig. 5. Errors of the implicit tree broadcast algorithm.

Nodes forwarding messages must know which subtree they are responsible for. Each message is therefore tagged with a small integer, which denotes the height of the subtree. This shows how many bits the address of the subtree should share a common prefix with the NodeID of the node the message is sent to. As the Kademlia protocol requires that at least one node is always known for every subtree [1]; there is no need to maintain an auxiliary routing table for the broadcast. Messages are forwarded to the subtree and all the smaller trees:

```
broadcast: function of(text, height)
  for i=height to number of bits
    if bucket i is not empty, then
      select any random node from bucket i
      send the message to the node: text, i+1
    endif
  endfor
```

This method is cost-efficient as there are no duplicate messages. Problems arise when there are packet losses on the network, as not only single nodes, but complete subtrees will miss the broadcast. Messages are practically directed to subtrees in this method: the original sender sends the message to the neighboring half tree, and is itself responsible for his own half tree. Then it sends to the other quarter of the overlay, and is responsible for its own quarter and so on. Every subtree has a single responsible node.

Fig. 5 was generated by our Kadsim application discussed in the next chapter, and it shows a simulation of this method. Nodes shown as white dots received the message, while black ones did not. As one can see, there are complete subtrees drawn in black. It is possible for such a message to be lost, which was sent to a high subtree. In a worst case scenario, the number of nodes not getting the message can be more than 50%, not even depending on the packet loss ratio. Although the network is decentralized, this algorithm is not, in its essence; as the importance of messages is vastly different, depending on which subtree they are addressed to.

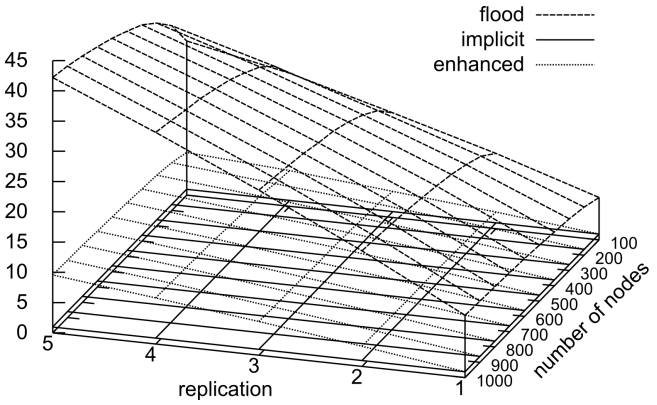


Fig. 6. Number of messages generated by each broadcast algorithms

Third algorithm: broadcast using the topology with replication. Addressing the problem mentioned above, the two algorithms can be combined. This algorithm is similar to the second, but from every subtree, not only a single, rather multiple nodes are selected to be responsible for forwarding the message. This way the probability of skipping a subtree is falling rapidly. Duplicate messages are possible in this case, so a unique identifier is required for all broadcasts initiated. Replication level can vary from 2 to k, the size of k-buckets.

4.1 Comparison of Broadcast Algorithms

To evaluate the algorithms presented above, we used our previously mentioned application specific simulator *KadSim*. The simulator models a moderately sized DHT network (in our tests with up to 1000 nodes). A Kademlia routing table is generated for the nodes at the simulator setup. Latencies are assigned to each node pair which communicate with each other. During the simulation, broadcast messages are randomly initiated by nodes, and are disseminated with the algorithms discussed above.

The simulator records the following data:

- number of all messages sent,
- number of messages per node,
- the number and ratio of nodes receiving the broadcast,
- time required for sending the message to as many nodes as possible.

In terms of traffic costs, flooding gives the worst results. The number of messages grows rapidly with increasing the node count or sizes of k-buckets. The second algorithm using the implicit multicast tree evidently results in one message for each node. For the third method the number of messages grows rapidly for large k-buckets, but only slowly for increasing the number of nodes. For $k = 5$, there were 7 messages/node for an overlay of 100 nodes, and only 9 for 1000 nodes (see Fig. 6.)

To evaluate the reliability of the algorithms, we simulated an overlay of 200 nodes. Packet loss ratio varied from 0% to 20%, replication from no replication to fivefold. Flooding almost always yields perfect results, due to the enormous number of messages. The reliability of the enhanced algorithm is of course the

same as the second for $k = 1$. In turn, using $k = 2$, this algorithm produces 90% reliability even for one fifth of the packets lost; $k = 3$ gives 97%.

Simulating the first (flooding) algorithm gives us the shortest time, in which the broadcast can be achieved, given an overlay and its routing tables. Many P2P systems select nearby nodes to speed up lookup and other operations [2]. This can also be used in Kademlia, and can also reduce the time required for the broadcasts. The factor of the achievable speed-up depends on the exact distribution of round trip times (RTT's) in the network. Increasing replication can also be used to speed up the broadcast messages, as network packets can use links with smaller latency.

4.2 Applicability of Broadcast Messaging

DHT networks have a well-known drawback, namely that they do not support partial keyword searches. This is due to the properties of hash functions – a very small change, even one missing character from the filename causes the hash value to be totally different.

Broadcast messaging can be used in DHT networks to implement partial keyword searches, for example, part of a filename. In this case, the lookup is not based on the hash function, but the value of the key itself. A lookup for a file in this case works similarly to that used in the Gnutella network. By using the inherit topology of the network, the number of messages used for the broadcast can be limited and thus can be much more effective.

The P2P network named BubbleStorm uses broadcast messages to lookup data stored in the overlay created by its nodes [8]. BubbleStorm replicates pieces of data and also search queries to many randomly chosen nodes, in the hope that the set of nodes storing data and the set of nodes receiving the query will intersect. The advantage of the Kademlia network with our broadcast algorithm compared to BubbleStorm is that the latter generates high network traffic for file store requests as well, and therefore it is not as efficient as Kademlia. Furthermore, BubbleStorm is not based on a DHT network, and it must use broadcast messages to implement lookups of precisely given keys, too.

The use of broadcast messaging in our P2P based network intrusion detection method named Komondor is essential [10]. In Komondor, network hosts to be protected create a DHT overlay which is used to store data of detected intrusion attempts. The overlay enables us to collect and correlate intrusion detection data globally. If a possible intrusion is detected, an alert is sent to all nodes in the system. Komondor is also the reference implementation of the broadcast algorithm presented in this article, as the alerts discussed are sent via the broadcast algorithm described above.

5 Conclusion

The reliability of the two elementary services of the presented network-based intrusion detection system, namely sending attack reports and broadcasting alerts can both be increased using replication. The only system-wide configuration parameter affecting the substrate peer-to-peer network, the level of this replication, can be determined in advance, with the methods presented.

The described method can be applied as a substrate overlay for specific applications. The advantages of the developed method has been proved, however its implementation in a distributed network-based intrusion detection system needs further investigations of the stability of the system in case of churn network.

References

1 **Maymounkov P, Mazieres D**, *Kademlia: A Peer-to-peer Information System Based on the XOR Metric.*, Proceedings of IPTPS02, posted on March 2002, DOI 10.1007/3-540-45748-8_5, (to appear in print).

2 **Ratnasamy S, Francis P, Handley M, Karp R, Shenker S**, *A scalable content-addressable network*, Proc. ACM SIGCOMM 2001, posted on August 2001, DOI 10.1145/964723.383072, (to appear in print).

3 **hua Chu Y, Ganjam A, Ng T S E, Rao S G, Sripanidkulchai K, Zhan J, Zhang H**, *Early Experience with an Internet Broadcast System Based on Overlay Multicast*, Carnegie Mellon University, Dec. 2003.

4 **Bhagwan R, Savage S, Voelker G M**, *Understanding Availability*: *Peer-to-Peer Systems II*, Lecture Notes in Computer Science, vol. 2735/2003, 2003.

5 **Chu J, Labonte K, Levine B N**, *Availability and locality measurements of peer-to-peer file systems*, ITCom: Scalability and Traffic Control in IP Networks **4868** (July 2002), DOI 10.1117/12.475282.

6 **Czirkos Z, Hosszú G**, *Reliability Issues of the Multicast-Based Mediacommunication*, Margherita Pagani, Encyclopedia of Multimedia Technology and Networking. Information Science Reference, 2008, pp. 1215–1223.

7 **Rhea S, Geels D, Roscoe T, Kubiatowicz J**, *Handling Churn in a DHT*, Proceedings of USENIX Technical Conference (June 2004).

8 **Terpstra W W, Kangasharju J, Leng Ch, Buchmann A P**, *Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search*, SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications (2007).

9 *P2P-Next Project: Shaping the Next Generation of Internet TV*, May 2009.

10 **Czirkos Z, Hosszú G**, *Peer-to-Peer Methods for Operating System Security*, Encyclopedia of Networked and Virtual Organizations (Putnik G D, Cunha M M, eds.), Idea Group Inc., 2008, pp. 1185–1191.

11 **Douceur J R**, *The Sybil Attack*, Peer-to-Peer Systems, Vol. 2429/2002, Springer, Berlin / Heidelberg, 2002, pp. 251–260, DOI 10.1007/3-540-45748-8_24, (to appear in print).

12 **Benayoune F, Lancieri L**, *Models of Cooperation in Peer-to-Peer Networks – A Survey*, Universal Multiservice Networks **3262/2004** (2004), 327–336, DOI 10.1007/978-3-540-30197-4_33.