Novel Applications of the Peer-to-peer Communication Methodology

L. L. Tóth, Z. Czirkos, G. Hosszú, F. Kovács
Department of Electron Devices
Budapest University of Technology and Economics
Email: hosszu@eet.bme.hu

Abstract: **In this article we represent two novel applications that make extensive use of the peer-to-peer communication method. Dolphin is a file sharing system with improved reliability and searching efficiency compared to other popular file sharing applications. Komondor is a peer-to-peer based security application, a network intrusion detection system that relies on the collaboration of network hosts. These two applications base their speed, stability and robustness on the application level network they create.**

Keywords: Peer-to-peer, P2P, overlay network, metadata, file sharing, reliability, collaboration, search method.

I. INTRODUCTION

The Internet is a shared resource and a *cooperative network*, which is composed of millions of hosts around the world. Nowadays, people use ever more applications that are capable of using the network [6].

Besides antivirus applications, file sharing utilities are the most popular type of software, which is downloaded by the users of the Internet at most. No surprise that a significant part of the Internet traffic is generated by *peer-to-peer* (P2P) services. The main purpose of the file sharing systems is that they search files shared by the users on the basis of some algorithm and also manage the downloading of them.

In this article we represent *two novel peer-to-peer applications*, which make extensive use of the P2P communication model. The first one is *Dolphin*, a community builder application that uses metadata-based keyword searches for files. The second one is *Komondor*, which is a network intrusion detection and prevention system relying on the collaboration of participants.

When designing *Dolphin*, we focused on the search based on the metadata of the shared files, which notably reduces the time of the search. Metadata is structured information that describes, locates, and makes it easier to retrieve, use, or manage an information resource. Metadata is frequently called data about data or information about information [7].

Our *Komondor* project focuses on the stability of the network used to share information about intrusion attempts and other suspicious events. The main goal of the design is to create a P2P network being robust enough to deal even with attacks targeted directly against it.

II. BACKGROUND

Components of the application level networks (ALN's) are the services operating on the different levels of the network, which control the communication session of certain networked application with its application level protocol. Sometimes they are called *overlay networks*.

The creative individuals of the ALN are called nodes or peers because they are responsible for building and maintaining the network equally. They can either be servers, when sharing resources with others or they can be clients as well [2].

A. The software model of the overlay

In contrast to common knowledge, peer-to-peer is a technology, not an application. Usually P2P functionality is only a part of the application, like the graphical user interface or the database functionality [5].

Real P2P networks and applications are distributed systems without any centralized control or hierarchical organization, where the software running at each node is equal in functionality. A review of the features of recent P2P applications yields a long list: redundant storage, permanence, selection of nearby servers, anonymity, search, authentication, and hierarchical naming. Despite this rich set of features, the core operation in most P2P systems is efficient location of data items [10].

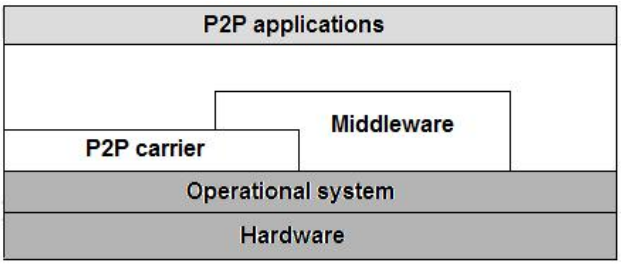

Figure 1: Model of P2P applications

Fig.1 shows to architecture of P2P applications. The task of the P2P carrier is the overlay creation, administration and file localization. The middleware maintains an assistant scope of duties, for example the selection of correct peers based on their distance and network link quality (the correction of P2P performance).

B. Centralized overlays

As a start of the network-based collaboration in 1996 the application SETI@Home was launched. It is a scientific experiment that uses Internet-connected computers in the Search for Extra Terrestrial Intelligence. SETI@Home distributes a screen saver based application to users that engages various signal analysis algorithms to process the centrally distributed data of radio-telescopes. At the time of writing, it had signed up more than three million users and had used over a million years of CPU time. The client software contacts a server in order to download data for processing and then processes them until the problem is solved and after sends the results back to the server. Should this processing fail, the data segment is assumed to be lost and therefore is ignored [11].

Napster was another one of the first P2P systems, as an example of a so-called hybrid system. Napster's file sharing is decentralized: one Napster client downloads a file directly from another Napster client's computer. At the same time the directory of files is centralized, with the Napster servers answering search queries and brokering client connections. This hybrid approach scales well: the indexing of files is efficient and uses low bandwidth, and file sharing is carried out *on the edges of the network* [6].

The advantage of such architecture is that it is fast and efficient until the capacity of the service server is reached. The search requests require relatively small network traffic. The bottleneck of Napster is however the central indexing server. The most serious shortcoming is that the working of the whole system will stop when that server fails [3].

C. Decentralized, non-structured overlays

The next generation of the P2P systems is *distributed, homogeneous, 'true peer-to-peer' architecture* [5]. Its most known example is the Gnutella network.

Gnutella was one of the first decentralized technologies that would reshape the Internet and reshape our way of thinking about network applications. The traditional knee-jerk reaction to create a hierarchical client/server system for any kind of networked application has been reconsidered. Decentralized technologies have many desirable qualities, and Gnutella was an excellent proof that such technologies, while young, are viable [4].

In such a system, every user's application is indeed a peer: *the nodes have completely same function*. In Gnutella, the data of a shared document is not on the main server. Every participant knows which attributes belong to the documents stored by him. The search

request for documents becomes more complicated: the search of the given file in the network is not limited just to the transfer of the data to the main server. One of the nodes gets a request; then it checks if the given document can be found there. If not, it sends it further to the nodes recognized by him. Those also make the same operations concerning the request.

The merits and the shortcomings are clearly visible. The failure of one of the participants in this system or the shutdown occurred from some other reason does not cause any problem in the operation of the whole Gnutella overlay. However the search will be slower and it will generate higher traffic on the network. If we want to have a greater chance of finding the document by the resending the request to more nodes, we should consider the fact that the node chosen by us may not tolerate the network traffic needed for that.

D. Decentralized, structured overlays

In structured overlays, documents and other resources are stored in a *well defined place*, so all peers know exactly where it can be located [2]. The advantage of these systems is the quick resource lookup that generates only minimal network traffic. Their shortcoming is the decreased flexibility and also their complicated architecture. Keyword lookups, overlay setup maintenance and other services typically require complex algorithms. An additional problem of these is the churn of participants – i.e. the continuous variation, login and logout of nodes.

Structured P2P networks assign unique identifiers, called NodeID's to participating nodes, and store key-value pairs. Every piece of information has a key, for example the name of the file. This key is scrambled with a hash function [18], which generates a seemingly random number derived from the key. Then these numbers are used to assign files to specific nodes: each node stores the files that have their names' hash value numerically close to the unique identifier of the node. That is why structured networks are also called *Distributed Hash Tables* (DHT's), as for every key, it is easy to find the node, which stores the corresponding value. This process is called consistent hashing [10], as every node uses the same hash function.

Examples for structured networks are CAN [17], Chord [10], Pastry and Kademlia [16]. These are all DHT's, but they use different topologies and routing mechanisms. In this context the routing means the forwarding method of the lookup request on the overlay.

E. Distributed hash tables

In a distributed hash table, a *hash function* is used to derive a small number from a key representing some data (the value). Information is then stored on one of the participating peers. The *application level networks* mentioned above all use hash tables to store information, but the exact management of storage is different due to the following three aspects (see Table I):

TABLE 1: TOPOLOGIES OF DHT NETWORKS

Overlay network	Metric	Topology
CAN	Euclidean	n-dimensional torus
Chord	Subtraction	Ring
Kademlia	XOR operation	Binary tree

- The selection of a *hash function*. Most networks use MD5 or SHA-1. The choice of a function is not really important, as they are only used to make data evenly distributed among the nodes. For example, Kademlia uses SHA-1 [14].
- The selection of a *metric function*. A metric defines a distance between two ID's, so hashed data can be assigned to specific locations, nodes in the network. Every node stores key-value pairs having a hash value closest to its ID, according to the metric function. Kademlia uses the XOR operation.
- The selection of a *topology*. This is closely related to the selection of the *metric function*. CAN is

usually symbolized as an n-dimensional torus, since it calculates the distance between identifiers by using the Pythagorean Theorem. Kademlia is usually represented with a binary tree. In Chord, the peers are organized in a ring, and messages are always sent clockwise the ring.

III. OUR NOVEL COMMUNITY SYSTEM

In a university, usually smaller groups would like to exchange useful files – sometimes even in places, where there is no accessible Internet infrastructure (for example students travelling together by train). In this case it is not possible to use heavy-duty, factory-made file sharing software, because those are inoperable in such situations. In contrast, our software is reliable even under such circumstances [12].

The two main expectations, which arise from these circumstances are:

- The system must work when the overlay fails and even when an *ad-hoc* network with a few users occurs.
- The file registering will be suitable for the claim of the student.

According to these requirements, the software developed by us uses the following two methods:

- *Metadata based file searches* – it extends the efficiency of document lookups.
- *Extends the maintenance of the network* – reliability during overlay failure.

We developed these two methods in our application. At the conditions mentioned above, our method is working reliably; it is also simple, supposing to have a few clients. The developed method (which got the name 'Dolphin', considering that it is reliable, handful and user friendly) is vastly different from large file sharing software, since the properties listed above especially support the cooperation in the small community.

To test our proposed solution for the problems mentioned above, we developed a P2P file sharing application. It allows sharing and exchanging of the files by university students regardless of which profession they study. It can use both centralized and decentralized structured overlays, so it may be categorized to be somewhere between 'Napster' and 'Gnutella'.

During the designing our main goal was the reliability. Namely, if any peer loses its connection to the server, but has a network access, the system should stay available. If connection is lost, it stores the IP addresses of other nodes in a file, and starts sending ping messages to them. If it gets an answer from somebody it can send messages further. This event is logged in a file that contains information about system errors, which has been sent to the server after the system recovery, so we can check the incidental problems in the future.

Our new function in Dolphin is the capability of searching files by metadata allowing *fast lookups*. This information contained by metadata we call a content-package. For example, in the library the register contains the main information about the book (writer, title, publication date, genre, place of storage, date of rents, etc.). The data of register refers to the data of the book.

There is a database on the main server, in which we can store the data and the place of the sharing files and also those of the file owners, too. It can store and check all traffic information, and also error reports. As in the Napster model, peers getting file information can connect directly to each other. The file transfer itself happens without the help of the main server.

Dolphin's exclusive purpose is the sharing of training supplements (lectures' materials, drafts, old tests, exams, home papers), especially documents (*.doc, *.pdf, *.jpeg, *.bmp, *.zip, *.rar, etc). It does not support the sharing of any other music or video files. Until now we tried to solve this by limiting the size of the documents shared, assuming that the size of the documents targeted for sharing is far below 10 MBytes. By this we excluded the barter of movies and applications from our system, since a lot of file share

programs exist nowadays, which can be used for the transaction of larger files and Dolphin is not really needed for that purpose. The size of the music files is usually under 10 MBytes, but the sharing of these is currently restricted by system. Certainly, only those materials are meant to be present, which were downloaded exclusively by their owners and are allowed to be copied in future. We should consider legality issue too in order to avoid the illegal file transaction.

If a given peer logs-out or is disconnected from the server, the system will mark the files belonging to him and during the search it will be visible, that the peer is off-line and that is why the files cannot be shared.
Registered users can even chat with the help of built-in chat program. The user should accomplish further operations while providing files meant for sharing:

b. Filename and reaching track should be provided
c. Metadata should be also added to the file

If after the logging in the connection with the server breaks, then the active IP addresses downloaded during the previous logging in are listed from a file and search requests are being sent to these IP addresses.

IV. THE EVALUATION OF DOLPHIN AND FURTHER DEVELOPMENTS

For the proof of concepts presented above, we developed the reference implementation of Dolphin in the Object Pascal language, in Delphi. The components Delphi provides enabled us to focus on the realization of our P2P methods, as other usual tasks (e.g. database connection, file downloads) are already built in into the development environment as components.

We want to control the reliability of the designed method with the further survey, besides we plan to build in some new functions too. Such new function would for example be the quick virus control before the data downloading.

A. The server and client functionality

If the main server is available, simple server–client architecture evolves. Here the documents are easily distributed: the clients are managing the human–machine interface, file sharing, chat, searching, downloading and personal data. The server manages the clients, generates the appropriate SQL commands. The MySQL server maintains the database of documents and storage.

If the main server is not available, a client takes over its role. It reveals its IP address to the public so other clients can join it and download requested files. This function comes in handy when there is no Internet connection, but the users want to share files.

It may seem obtrusive that there is a separate query and an executive order, since both of them are just executive commands. However, in the case of query the returning data should be managed. This can be helped by the data source component, which is in the close connection. The returning data appears directly in the data source.

B. The set-up of the client

It clears up from the structure of the client that incoming data is converted by data-record formation; then depending on the control it goes towards server or into downloading control. The transfer of the files is achieved by the Delphi TServerSocket and TClientSocket, both designed to let you read and write information over a TCP/IP connection [9].

C. Function of the client

The client disposes some separate functions, which should be represented individually. Here are these grouped:

1. Maintenance of the data
   a. *Registration*: The admission of a new user and automatic logging-in after.
   b. *Logging-in*: The activation of an existing user in the group, which signals that the downloading of his files is possible.

c. *Logging-out*: The inactivation of the online user.
d. *Erasure of the registration*: With the help of it not just the user disappears from the group, but the list of the files shared by him does the same!

2. Maintenance of the shared files
   a. *Modification of the file-list*: Fixation of the access path and metadata, deletion of the file sharing; it is important, that during the admission of the new files the user must give the pop-up list of the metadata.
   b. *Update*: It makes the local existing file-list accessible even in the database so the efficiency of the search is also raised.
3. Search
   a. *Search*: It makes it possible to search the files by their metadata. Applying the metadata in the searching process a content-based lookup method can be obtained.
4. Chat
   a. This function is completely separate, whereby active users can communicate with each other in writing; there is the possibility of creating the friends-list, with the help of which the user can mark admitted users and check every minute, which of them is active. This chatting includes many useful features, one can chat not just with the users, who are on the friends-list, but also with those other members, who are active [5].
5. User's control
   a. Every such element, which can influence the functioning of the program, belongs here. Practically speaking, the user can see just that from the program.

D. Future trends of Dolphin

The Dolphin system is not just a file-sharing service, but also community building software, in which we would like to build in the functions listed below. The users can leave each other a message (mail) with an easy basic message sending functions. The messages are stored by the central server for the definite time if the recipient is not available and if the recipient peer logs in, the server deliver the message to him. We also plan to build in some other functions, with the help of which peers can exchange some useful information (forum), mail-lists, advertisements, news, job offers, student jobs, University parties, radio and flashers, which will appear on the header of the client program as a flash messages. We also want to create some collection of the more important study links.

Should a peer overload the server (for example, the vicious attack, denial of service), the server will automatically break off the connection (approximately in 10 minutes). The database will record if there was a problem with a peer and thus later it can be investigated what happened [1].

During the future developments we would like to focus on the deeper research of the Dolphin application. Our plans include the supply of the graphs, which will show the performance of the system. Furthermore we are going to improve our method by implementing the known TCP traffic based analysis method for handling the possible failures of the peer-to-peer traffic [8].

V. REALIABILITY OF PEER-TO-PEER SYSTEMS

Reliability of a P2P network is directly *influenced* by the dependability of the underlying network packet transfer service. The different topologies, however, are affected differently by packet losses and other errors.

Most structured P2P networks, for example CAN [17] have an exact topology. In the n-dimensional circular torus of CAN, every node has to maintain only a small number of connections; in a 2D example, this number is four (up, down, left and right). Data to be sent is forwarded (routed) on the overlay network from node to node, finally arriving at its destination. CAN is therefore able to use session-oriented TCP as its transport protocol, as a node always communicates only with its neighbors.

There are other structured networks, which have *no specific topology*, for example Kademlia [16]. In Kademlia, messages are not forwarded inside the overlay (there is no routing defined), rather they are sent directly between the source and the destination as datagrams. The purpose of the overlay is only to find the physical network address (IP address, port number) of the destination node in question, and it uses UDP for its messages. Therefore network errors directly influence the communication between peers, and this is especially true for specific source-destination pairs. Permanent network errors, nodes that cannot be reached (because they are behind a firewall, for example) all degrade the quality and performance of the overlay.

The availability of a specific connection can naturally be tested by a simple ping message. Due to network errors, information available at nodes *can sometimes be unreachable for others*. The exact distribution of errors is usually highly uneven; with some nodes having good connectivity and others not. This issue can be solved by the data replication. As node ID's are usually chosen randomly, nodes which are close to each other in the application network address space can be quite far from each other in the physical address space, and even geographically. Therefore sending messages to more than one node, which are close to a specific destination can result in replicating data at very different locations; almost as if destinations were randomly chosen.

Fig. 2 shows our *simulation* of a random Kademlia overlay topology. The simulated scenario was that we tested all the participating nodes of the overlay (in this experiment we used n=200 nodes) if they are able to send an information message to a certain destination node and some replication of the information message to the closest nodes of the destination.

We did not use a real topology, since in Kademlia there is not real topology, in fact every node can send message to every other. But the success of the sent message was measured by a random variable, namely the ratio of the bad links of each participating node. In this experiment, a replication factor of k=8 was selected, which means that every participating node stores its key-value pairs at eight different locations.

The ideal case is when all network connections are functioning, and there are no errors. If there are failing connections, senders of messages choose nodes as destinations, which are not the eight closest ones in the entire network, but a bit further in node identifier address space. This can happen since peers can detect the failing links. E.g. a node is intending to send its message to eight peers, but it detects that the node with the 3rd closest address is unreachable. Then it sends its piece of information to the 9th closest node, too.

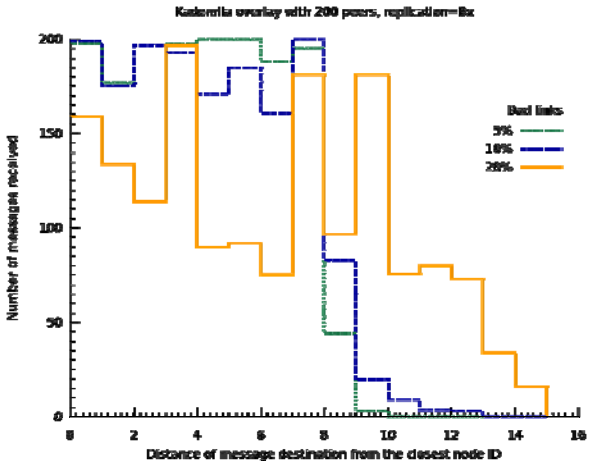

Figure 2: Simulation of connectivity in a Kademlia overlay

On *Fig. 2* the three different plots visualize network messages arriving at destination nodes, in case of various ratios of non-functioning network links. The X axis shows the nodes of the overlay, they are sequenced in the order of closeness to the destination address. X=0 is the closest one, the higher the sequence number on the X axis is, the further the node denoted by the actual sequence number is from the destination node. In other words, X=0 is the primary destination (closest), X=1 is the secondary (second closest), X=2 is the tertiary and so on. The Y axis shows the number of nodes, which could reach the destination. This is the value which is our point of

interest. Let us suppose that we have a key-value pair stored in the overlay with 200 nodes, and 20% of the links are failing. The information is stored at different locations. We are willing to find a node, which is accessible to almost all the nodes, so anybody can retrieve the key-value pair. This was the fourth node on Figure 1 for the 20% error rate simulation.

As a result of the simulation presented on *Fig. 2*, even with high ratio of bad links in the overlay network, the content replication helped to find a node, which can store the information of the message. With 20% of failed links, the node that was only the 4th closest to the original destination (see the horizontal axis on Fig. 2), could still receive messages from all other nodes dependably. In case of high number of such experiments we found very similar results.

A. Peer-to-peer overlays in real-world networks

P2P networks are usually not that effective on the Internet, as in a computer laboratory experiment. One of its reasons is that many nodes participating in the network are behind NAT (network address translation) or firewalls. Those usually cannot receive incoming connections, but are only able to initiate outgoing ones [15]. That is a serious drawback in peer-to-peer applications, where the equal role of the participators is a fundamental requirement. Another important fact which has influence on the operation of an overlay in a real-world environment: a node inside the network can have good or bad connectivity due to the random network errors. The distribution of network errors is similar to that in a power-law graph [13]. In the following simulation this fact is used.

As mentioned previously, Kademlia does not have an exact topology. To store data in the overlay, a node is required to send FIND_NODE messages to nodes *successively closer to the destination identifier* (see [16] for an explanation of Kademlia routing messages). The replies for these messages are IP address, port number pairs; network addresses of other nodes [16]. Finally, the node intending to store a data finds out the Internet address of that node (it will be the destination node), which has its application level address the closest to the hashed key, and sends a STORE request directly as a datagram. Due to the inherent network errors the data will be sent not to the destination node, but to some other nodes close to the destination, depending on the actual value of the replication.

Unlike other overlay networks like CAN, which require nodes to maintain only a small number of connections to neighbors, in Kademlia overlay, a node (called peer) can send a message to anyone. If we have e.g. n=200 nodes in a Kademlia overlay, it is possible for a node to send a message to any other participator peer, however, it cannot know for sure, that its connection to the destination address will work. The node will get the network address of the destination, but maybe it cannot connect it, for example because of firewall settings. This is a problem, as data stored far away (in *nodeID* address space) from the normal destination cannot be retrieved by others: they will not know that they should request that piece of information from that node. The most fundamental assumption for a distributed hash table was that every key-value pair is stored in a well defined place.

We modeled this distribution with the number of permanently failing links increasing quadratically, based on the power-law graph model [13]. The ratio for a given peer is then given by:

$$h(m) = c \cdot \left(\frac{m}{n} \right)^{\alpha} \qquad (1)$$

where m is the sequence number of the examined peer, n is the number of all peers ($0 \leq m < n$), α is 2 for a quadratic distribution. c is a constant, which sets the maximum proportion of errors (ratio of inaccessible neighbors for a specific node). These values can be set experimentally, and depend on the size and the actual properties of the Internet network underlying the Kademlia overlay. Local area networks are usually much more dependable, than wide area networks like the Internet.

This h(m) function gives the ratio of bad connections for a given peer in case of n=200 for various α and c values. The X axis shows the parameter m, the Y axis shows the h(m). The function h(m) should be a stepping function, as the result multiplied by the number of nodes gives the number of bad connections, and the h(m)*n product should be an integer, since it is the number of the bad links. Therefore this formula is estimation, as one cannot interpret '0.5 links fail', only 0 or 1, which will be very inaccurate. On the other hand, rounding '10.3 links fail' to 10 failing links is only a small error. Fortunately, we are interested in modeling the operation of the overlay in a heavily error prone environment; the final equation derived in this paragraph is not applicable for a low number of errors. Remember that as STORE requests end up at destinations, of which nodeID's can be taken into account as random variables, thanks to the properties of hash functions, it does not really matter, which error link ratio belongs to which node. Only the global distribution of the node peers is important, that some nodes can receive most of the messages, some not.

As the underlying Internet network is not perfect, we also cannot expect the overlay to be so. But still we can have a requirement, expressed in numeric terms, for example 99% of all the cases we should be able to retrieve the stored key-value pair from the Kademlia overlay. Similarly, the original Kademlia paper [16] gave a probabilistic guarantee for a key-value pair being available for lookup over time.

We have the ratio of allowed errors (β=1%), and for a lookup to be successful with a probability of 1-β, the inequality should hold for the given node, which is responsible for storing the key-value pair in question, and able to answer the request. Due to the fact that the return values of hash functions seem to be random variable, and the probability distribution of this random variable is practically equal distribution, (2) must be valid for every m value in the interval [0, n]. That is why we can choose m freely, so m/n is virtually a random number between 0 and 1. Also, nodes chose their identifiers (nodeID) by virtually picking a random number in the address range (due to the properties of the hash function), so this way the stored data always gets to randomly chosen hosts, at least we can suppose it in the terms of simulation and modeling.

$$h(m) \le \beta \tag{2}$$

If we solve (2), we get the ratio of nodes, which fulfill our requirements accruing to the allowed error ratio β. The solution of (2) if the (1) is substituted is:

$$\frac{m}{n} \le \alpha\sqrt{\frac{\beta}{c}} \tag{3}$$

The right side of (3) can be interpreted as a condition which must be fulfilled. If it is, a certain piece of information stored in the overlay can be retrieved successfully, too. We denote the probability that the lookup procedure is successful with P'. Since $0 \le m/n < 1$, and randomly changes from 0 to 1 (virtually, due to the hash value), the following equality holds for P':

$$P' = \alpha\sqrt{\frac{\beta}{c}} \tag{4}$$

If the Kademlia overlay implements replication k, it has more than one, exactly k opportunities to store or retrieve data. Practically speaking, it can choose more than one random m number, and the probability of correct lookups denoted with P increases. Calculating the probability of all lookups failing, and then subtracting that from one, we get:

$$P = 1 - (1 - P')^k \tag{5}$$

which gives the probability of successful looking up a given information despite network errors. In this formula, k is the level of replication, the number of nodes storing a given key-value pair.

Equation (5) can be used to estimate the necessary replication factor k, if the ratio of network errors and required probability of correctness is given. Fig. 3 shows the results for given error and replication levels, with 1% failure allowed. As one can see, for *h(m)=10%* of failing links for example, replication factor of *k=4* is enough to ensure correct operation

with probability $P>0.6$. This replication factor k is essentially the same as the size of the *k-buckets* in Kademlia; see [16] for a discussion. The model we presented here 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.

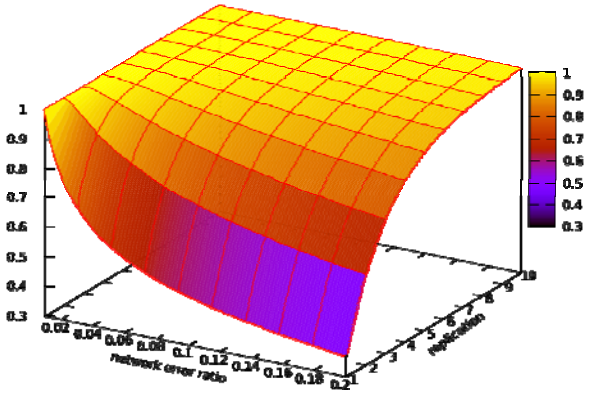

Figure3: The probability of correct lookups in Kademlia.

VI. THE NOVEL INTRUSION DETECTION SYSTEM

Our novel network intrusion detection and prevention system, *Komondor*, is built on top of a distributed hash table. The inherent stability of such an overlay network enables it to work even when a significant number of network links or nodes fail – which might be a common scenario in the life of an intrusion detection system, as an attacker might try to stop the detection system before conducting his real attack.

We have chosen *Kademlia* to be the substrate of the detection system. The DHT method can be used here with the idea of IP addresses being keys and attack reports being values stored in the overlay. Every time a node detects a suspicious event, it hashes the IP address of the suspected attacker. This way, the report of the event is mapped to some other Komondor node, and sent to it. We call the destination node of the report the *collector node*. As every node uses the same hash function, *reports about the same attacker end up at the same collector node*, so that participant of the network has all information regarding the attacker in question. Analyzing the reports and seeing if the events suggest a real attack, it initiates a broadcast message over the network to notify all other participants about the possible danger.

This system also enables nodes to detect *network-sized attacks*. Experience suggests that a single lost connection is usually caused by a link failure or some software problem, but a big number of lost connections always suggest a network scan. These network scans, for example, are used by attackers to find weak points of the network. By collecting information from many detectors, these can be revealed with high probability.

VII. CONCLUSION

We have developed our file sharing and collaborating method called *Dolphin*. According to the carried out test runs, this novel method is able to support the reliable operation of the system including maintaining its overlay network even during the network errors and network partitions.

The *improved search method* is based on keywords, where the effectiveness of the searching is context-based. The advantage of this procedure is that the system can search for a required data based on the additional metadata. From the obtained results, according to our pensiveness, we experienced that in case of switching off the server the file search functionality remained available, however, the searching process naturally slowed down.

Contrary, when the server is available, the search is very effective. The improved metadata-based search does not need a powerful search engine that makes possible its use in a smaller community even on an ad-hoc network.

Our reference implementation of Komondor has been collecting data for years. Examining the database of detected intrusion attempts, we have concluded that this system can be highly *efficient against attacks which are deliberate and targeted*. Attackers who have a well-defined goal usually try to break into a system by multiple means; if any of the attacks'

manifestation is revealed, the whole network can be protected. For random attacks, i.e. virus software, other protection methods must be used.

REFERENCES

[1] Dingledine, R., Freedman, M. J., & Molnar, D. (2001). Free Haven. In A. Oram (Ed.), *Peer to Peer: Harnessing the Power of Disruptive Technologies* (pp. 102-121).

[2] Hosszú, G. (2005). *Mediacommunication Based on Application-Layer Multicast*. In Dasgupta, S. (Ed.), Encyclopedia of Virtual Communities and Technologies (pp. 302-307). Hershey, PA: Idea Group Reference.

[3] Jelasity, M. (2007). *Peer-to-Peer Systems and Gossip Protocols*. Retrieved November 14, 2008, from http://www.inf.uszeged. hu/~jelasity/p2p/index.html#1

[4] Kan, G. (2001). Gnutella. In A. Oram (Ed.), Peer to Peer: Harnessing the Power of Disruptive Technologies (pp. 62-80).

[5] McKeeth, J. (2003). *A Guide to Peer-2-Peer*. Retrieved November 14, 2008, from http://bdn1.borland.com/ article/borcon/files/3214/paper/3214.html

[6] Minar, N., & Hedlund, M. (2001). *A Network of Peers: Peer-to-Peer Models Through the History of the Internet*. In A. Oram (Ed.), Peer to Peer: Harnessing the Power of Disruptive Technologies (pp. 8-19).

[7] National Information Standards Organization (NISO) Press (2004). *Understanding Metadata*. Retrieved November 14, 2008, from http://www.niso.org/ publications/press/UnderstandingMetadata.pdf

[8] Soldani, C. (2004). *Peer-to-Peer Behaviour Detection by TCP Flows Analysis*. Retrieved November 14, 2008, from http://www.run.montefiore.ulg.ac.be/~soldani/ P2P_Behaviour_Detection.pdf

[9] Zarco, G. (1998). *Exchanging Data over the Network using Delphi*. Retrieved November 14,2008, http:// delphi.about.com/od/networking/l/aa112602a.htm

[10] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: *A Scalable Peer-to-peer Lookup Service for Internet Applications*. Retrieved November 14, 2008, from http://www.sigcomm.org/sigcomm2001/p12-stoica.pdf

[11] Taylor, I. J. (2005). *From P2P to Web Services and Grids*: Peers in a Client/Server World. London: Springer-Verlag.

[12] Tóth, L.L., & Makkai, H. (2007). *Development and realization of new confidence and search methods of the Delfin file sharing system*. In Proceedings of the Scientific Students Conference, Budapest University of Technology and Informatics, Faculty of Electronics and Informatics Engineering.

[13] Albert, R. & Barabási, A.L. (2002). *Statistical Mechanics of Complex Networks*. Reviews of Modern Physics, 74, pp. 47-97.

[14] D. Eastlake, P. Jones. *US Secure Hash Algorithm 1* (SHA 1). The Internet Society, Request For Comments 3174, September 2001.

[15] P2PRG (2007). *The Internet Research Task Force Peer-to-Peer Research Group*. Retrieved from http://www.irtf.org, June 2007.

[16] P. Maymounkov and D. Mazieres (2002). Kademlia: *A peer-to-peer information system based on the xor metric*. In Proceedings of IPTPS02, Cambridge, USA, March 2002.

[17] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker (2001). *A scalable conten t- addressable network*. In Proc. ACM SIGCOMM 2001, August 2001.

[18] Eng Keong, Jon Crowcroft, Marcelo Pias, Ravi Sharma and Steven Lim. *A Survey and Comparison of Peer-to-Peer Overlay Network Schemes*. IEEE Communications, March 2004.

AUTHOR BIOGRAPHIES

Loránd Lehel Tóth received his M. E. from the Budapest University of Technology and Economics in 2009. His main fields of interest are peer-to-peer overlay networks and character encoding. He published many conference and journal papers as co-author in the field of the peer-to-peer based communication. He participated in the Conference of Scientific Circle of Students with the paper: "Development and realization of the new confidence and search methods of the Dolphin file sharing system" in 2007. He obtained professional experiences in the General Electric as a Purchased Material Quality Engineer Assistant, in Power Controls and Lighting division in 2008-2009.

Dr. Gábor Hosszú received the M.Sc. degree from Technical University of Budapest in electrical engineering and the Academic degree of Technical Sciences (Ph.D.) in 1992. After graduation he received a three-year grant of the Hungarian Academy of Sciences. Currently he is a full-time associate professor at the Budapest University of Technology and Economics. His main interests are the Internet-based media-communication and the multicast technology. He published several technical papers and wrote chapters and books in the field of the media-communication. In 2001 he received the three-year Bolyai János Research Grant of the Hungarian Academy of Sciences. He leaded several research projects. His current field of interesting is the different technologies of the multicasting, the peer-to-peer communication, collaborative security, and the VHDL based digital system design.

Dr. Ferenc Kovács received the M .E. from the Technical University of Budapest in 1959, the Academic degree Technical Sciences (Ph.D.) in 1981 and the Doctor of Hungarian Academy of Sciences degree in 2001.

He joined the Research Institute for Electronics in 1959 where he worked on the field of microelectronic design and test. Since 1982 he has been Associate Professor on the Department of Electronic Devices, Technical University of Budapest, getting the Professor title in 2001. Since 2001 he is also a Professor on the Peter Pazmany Catholic University, Faculty of Information Technology, Vice dean for the research field and head of the Jedlik Research Laboratory. He published more than 140 technical papers and seven books. His area of research are the development of integrated circuit test equipment, real-time applications of VLSI circuits, low-power design of CMOS circuits and the design of low-power biomedical instruments.

Zoltán Czirkos is a PhD student at the Technical University of Budapest. His main fields of interest are operating system security and peer to peer communication. In 2005, he participated in the Conference of Scientific Circle of Students, with the paper "Development of P2P Based Security Software", and received the second award. He published several technical papers and wrote chapters as co-author in the field of the collaborative security.