P2P based intrusion detection

ZOLTÁN CZIRKOS, GÁBOR HOSSZÚ

Budapest University of Technology and Economics, Department of Electron Devices
hosszu@nimrud.eet.bme.hu

Keywords: Peer-to-Peer, P2P, intrusion detection, NIDS, overlay

This paper presents a novel security method. The software entities utilizing this method create a peer-to-peer application level network, which is then used to share information about intrusion attempts detected. Data collected this way is then used to enhance the protection of all participants. The system is completely decentralized, thus it remains functional over an unstable network or when many peers are attacked at once. The Kademlia P2P overlay is found to be the most suitable to create such a network. The stability of the overlay and the broadcast algorithms are both analyzed in this article.

1. Introduction

A number of security systems are presented in the literature, which run instances of applications and different hosts which communicate among each other [3], [4]. The novelty of the solution developed by us is that its nodes create a *P2P* (Peer-to-Peer) overlay on the Internet. The organization is automatic and does not require any user intervention. This network model provides great stability, which is needed by the nodes to quickly and reliably exchange information. The system can remain operative even on unreliable networks due to attacks and link failures. The software that implements this solution is named *Komondor*, after a famous Hungarian shepherd's dog.

Section 2 of our paper presents related work and systems, which are similar to ours. P2P overlay networks are explained, and also two distributed intrusion detection systems are mentioned. Section 3 shows the design goals and internal operation of the Komondor system. Section 4 explains the Kademlia overlay network in detail, this is necessary to understand why it is the most suitable overlay to use as a substrate of Komondor. Section 5 summarizes our results and experience collected until now.

2. Related work

2.1. P2P overlays

Application level networks (or sometimes called *overlays*) with *peer-to-peer* (P2P) topology can be structured or unstructured.

The *nodes (peers)* of unstructured networks can easily be dispensed. The network handles joining and leaving nodes in a very flexible way. The usual queries (data lookups) are also processed by the nodes, forwarding the lookup query to each other. Examples for these unstructured networks are *Gnutella, Freenet* and *FastTrack*.

Structured peer-to-peer networks usually implement a *distributed hash table* (DHT). These networks store key-value pairs and enable the users to quickly lookup a value associated with a precisely given key. In contrast to unstructured networks, logical links between nodes are determined by a set of rules; the topology of the network is exactly defined. Every stored piece of information (or file) is sent to a precisely selected node. Nodes are assigned a *node identifier (NodeID)* chosen from a fairly large interval of numbers (for example, 160 bits).

Similarly, each stored piece of data is assigned a key, which can be a hashed value of a file name, for example. The output of the hash function is in the same domain as the node identifiers. Every node stores key-value pairs, which have their hashed keys closest to its own NodeId. So if one knows the key, it is easy to find the node storing the value associated with that key. This is called *consistent hashing* [8,9].

Structured networks are different from each other in terms of topology, routing algorithms and the distance function (which calculates the distance between two identifiers, or a hashed key and an identifier.)

2.2. Distributed intrusion detection

The usual distributed intrusion detection systems deployed on networks are centralized, and are generally used for collection of data only [4]. Applications which are decentralized and also capable of intervention (intrusion prevention) are presented only lately.

The prevention system named PROMIS (and its ancestor, Netbiotic) uses the JXTA framework to enable nodes to exchange information of detected intrusion attempts [12]. It builds an overlay network which is partially centralized. The nodes entering the PROMIS network receive information about the number and rate of suspicious events detected, and they tune the security level of the Web browser built into the operating system accordingly. This method gives a general protection against malicious applications, but also reduces usabi-

lity of the system. The approach is somewhat similar to the epidemic prevention measures used in daily life.

The spam (e-mail junk) filtering system named Spamwatch is built on the Tapestry network [13]. The application is a plugin for the e-mail client. The hashed content of the e-mail messages marked by users as junk mail are stored in a distributed hash table; the same message on another user's computer can automatically be discarded. By using the DHT, the lookup of a message is fast, and generates little network traffic.

3. The proposed system

In the Komondor system, detection of intrusion attempts is distributed, by means of a DHT's based on the Kademlia network [1]. The following goals were important during the development of the system:

- Building a stable overlay network to exchange information.
- The data should be exchanged as quickly as possible.
- Decentralization of the system, enabling nodes to be missing.
- Masking the security holes of nodes based on intrusion attempts detected.

The several hosts running the Komondor software create a virtual, application level network, which is sometimes called an overlay. The speed of exchanging data about intrusion attempts largely depends on the network model employed. The system is built on a peer-to-peer (P2P) based overlay to ensure decentralization and stability [11], in contrast to the client-server model with a much higher risk of failure.

In the Komondor system, a structured network, a DHT is employed to store data of intrusion attempts. Keys are IP addresses of intruders, values are the information of intrusion attempts. Report of intrusion attempts from a specific attacker will be sent to a single node, as all nodes use the same hash functions. If that node analyzes the reports and sees that the IP address in question belongs to an attacker, it initiates a broadcast message, to alert other Komondor nodes of the possible danger. Every node is interested in receiving information which enables it to strengthen its protection. Compared to PROMIS, the protection built up by Komondor is not general; rather it is against the recognized attackers only.

Detection at multiple points and collection of data can be very efficient. Consider the following example. Let us assume that an attacker is trying to find an open, badly configured SMTP server to send junk e-mail. It tries to connect to many nodes protected by the Komondor system to find out which nodes have an SMTP service at all. One node sees an incoming connection which is immediately lost. From the viewpoint of a single host, an event like this alone does not necessarily indicate an attacker. It could possibly be a really lost connection, caused by some link failure, or an e-mail sending canceled upon the user's request. But if this event is detected on many of the nodes, then it immediately becomes suspicious. In the Komondor system, the IP address of the attacker determines which node will be the collector of information. That node will be responsible for processing data about a specific attacker; so sharing information about every suspicious event is necessary.

The main goal of our research is the investigation of the stability and reliability of the *Kademlia* P2P overlay system, and also the exploration of possibilities of peer-to-peer based intrusion detection. Our Komondor software has Linux and Microsoft Windows versions currently implemented. It uses Snort and the system log files to collect information about events; to create protection, it tunes the firewall of the operating system. Other detection and prevention modules are also planned to be developed in the later versions.

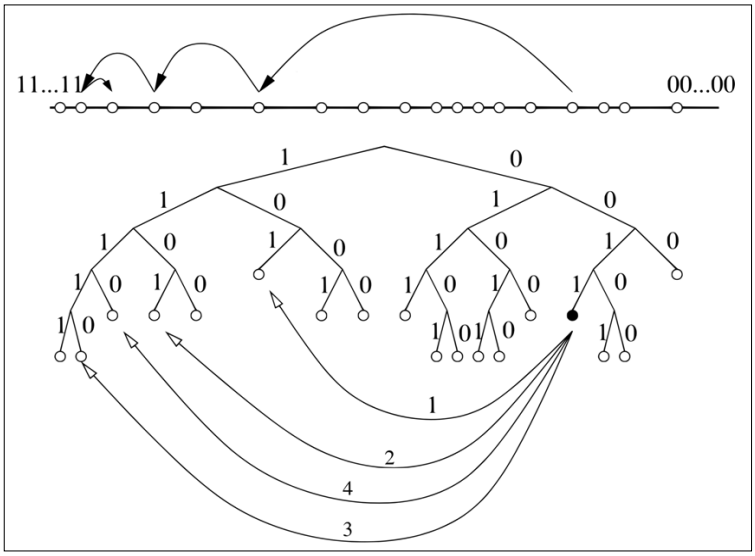
4. Using the Kademlia overlay in Komondor

4.1. The Kademlia overlay system

To investigate the stability and reliability of the Kademlia overlay, and to understand the broadcast messaging system built over the overlay, we outline the topology and internal operation of Kademlia in this section.

Kademlia uses distributed hash tables (DHT's). The nodes participating in a Kademlia network can be represented with a binary tree [5]. Kademlia nodes store the same amount of connection info (IP address, port number) for every subtree. These lists are called *k-buckets* in the original paper. The size of these lists is usually denoted by *k*, which is a system-wide configuration parameter. In a well populated network, taller subtrees usually contain a lot more nodes than *k*, so a node has rela-

Fig. 1. Routing in Kademlia



tively less information about the distant subtrees, as compared to the subtrees in close proximity.

Routing is shown on *Fig. 1.* (In papers about Kademlia, the subtrees which have only one node are usually not shown as a tree, only as a leaf. In the example below, all nodes have 5 bit identifiers.) If the node with the identifier 00110 would want to send a message to the node with ID 11100, it has nothing else to do that send a message to *anyone* in the 1* subtree, who will have greater knowledge of nodes in the 11* subtree and so on.

The order of lookups is shown by the arrows with numbers. The sending of message is completed in *O(log n)* steps this way. The distance of two identifiers is calculated using the *exclusive* or function. The magnitude of the distance between two peers is proportional to the height of the subtree containing both of them. This is why this network can be represented as a binary tree, and why it is called the *XOR topology*. Due to the symmetry of the XOR function, the distribution of incoming and outgoing messages is the same. The routing table of nodes is automatically refreshed by network traffic; so the network strengthens itself with all messages.

Comparing Kademlia to other DHT systems, its unusual property is the freedom of nodes (also requiring Kademlia to use UDP instead of TCP.) To lookup a value associated with the given key, the message is not forwarded from node to node, but rather a node itself finds out who the destination of the message is. This makes handling of replication very easy. A node intending to store a key–value pair does not send data to the closest node to the key; rather it sends it to the *k* closest nodes. By selecting a value for *k*, the stability of the overlay can be tuned. But as we will later see, by selecting $k>1$, the availability of data stored can also be enhanced.

The Kademlia protocol requires nodes to maintain lists of other nodes with at least *k* entries for every subtree. Connection information is refreshed in every hour, if necessary. The value of *k* must be chosen so that it would be very unlikely for all *k* nodes to quit the network in an hour. The nodes quitting the network are *not* required to send their stored key–value pairs to other nodes in the network. So if a node disappears, data stored by it would also be gone, if replication was not implemented. Note that in a DHT, being able to communicate with a node implies being able to retrieve key–value pairs stored by that node. So the level of replication must be the same as the number of nodes in a *k-bucket*. Thus this is the only system-wide configuration parameter needed by Kademlia.

4.2. Reliability of Kademlia

The overlay in the Komondor system is created by a version of Kademlia modified only slightly compared to the original. The conclusions presented in the following sections are also applicable to the original overlay. Our test runs of the Komondor system proved that replication in Kademlia is much more important than in other types of overlay networks, as it is very common in a real environment that nodes cannot connect to each other, due to packet losses, network address translation or other reasons. Therefore it is possible that a key–value pair stored by a single node cannot be reached by others, as some may not be able to connect to it. Replication partially solves this problem. If the pair is stored not only by a single node, but by a range of nodes (ie. *k* nodes), replication increases the probability that at least one node will be able to answer the lookup request. It is also possible that some k-buckets of nodes are not correctly populated with the addresses of other nodes at a time; replication in this case will also solve the problem of unavailability. (The routing tables of structured networks may be incorrect for small intervals, when a lot of nodes join or quit in a short time. This is called *high churn* [10].)

To prove the above statement, we developed a simulator application for Kademlia. It is called *Kadsim*. Although much of the simulations were focused on the Komondor network and its behavior, the results are general and can be applied to other Kademlia-based network, too. Kadsim works the following way. Given a number of nodes, it creates a connectivity matrix, which is essentially the adjacency matrix of the possible communication between nodes.

Also given a message, which is virtually a randomly chosen identifier; in the Komondor system, a hashed value of the attacker's IP address. The most important demand of Komondor against the overlay is that there should be always at least one node, where reports about a specific attacker can be collected. So Kadsim models the case when *all* nodes in the overlay detect some attack from the IP address in question. Every node hashes the IP address, and looks up the resulting identifier in the overlay. Nodes that cannot be reached by some other nodes do not count. (However, they may be reached by others.) Usual DHT networks, storing files for example, work exactly the same way; a given key is looked up in a tight range of nodes near the identifier.

Finishing the simulation, Kadsim sorts the number of messages received by each node, using the distance of NodeID's for the comparison. The results are plotted in *Fig. 2.* Ideally, when there are no network errors, and all links are operational, the resulting function is a single step: the *k* closest nodes to the key get all messages, and others get no messages at all. If there are link failures, the function will be lower and wider (see Fig. 2). For example, if the level of replication is $k=16$, and one of the nodes cannot access other nodes who are the 12th and 15th closest to the key, it will store key at the 16th and 17th closest ones.

If the distribution of link failures is flat, there will be no node in the overlay, where reports of attacks can be fully collected, no matter how high the level of replication is. In such a case, Kademlia would be a very bad choice for overlay topology. Real networks like the Internet are fortunately not like this: link failures are unevenly scattered throughout the network. There are hosts,

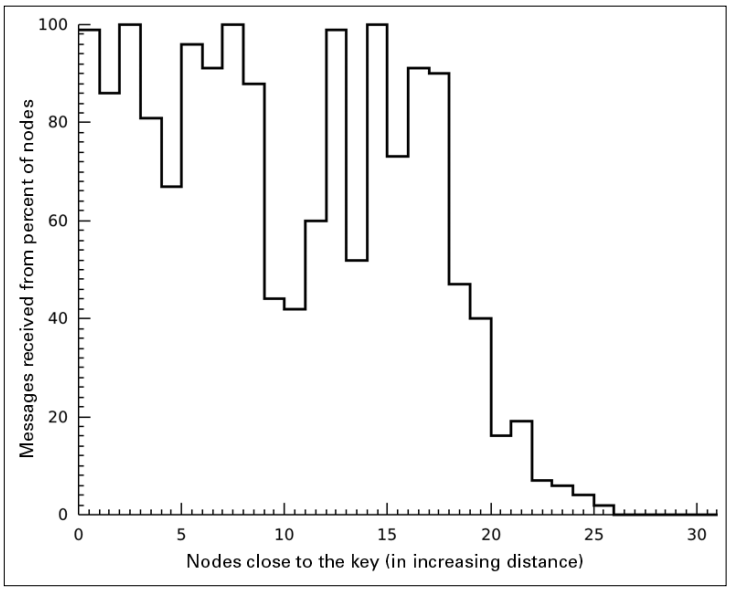


Fig. 2.
Storing keys in a Kademlia overlay; replication is 16-fold, ratio of failing links is 20%

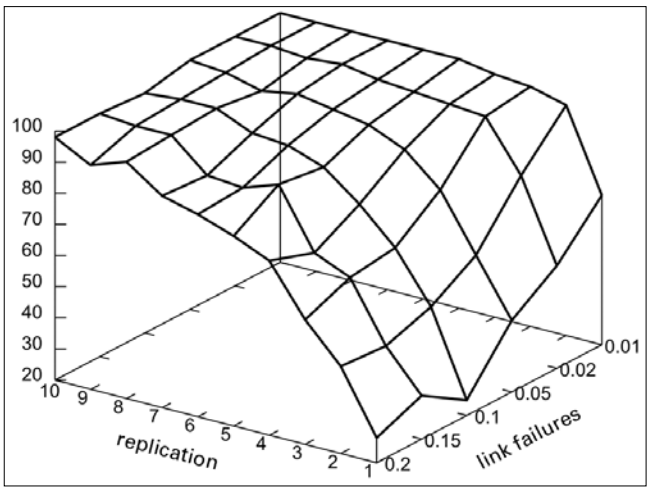
which have public IP address, those are easy to connect to. Others are behind network address translation, and cannot always be reached. The exact distribution can be very different for various networks; Kadsim models the distribution as a polynomial function. When this distribution is not flat, there are nodes who can receive the attack report with very high probability.

Simulation showed that a moderately high level of replication, *k=8* sufficiently ensures that a suitable node will exist in the network with high probability *(Fig. 3)*. For a hundred of nodes this seems to be too much, compared to other P2P networks, but increasing the number of nodes *there is no need to increase replication.* There will be at least one of the selected nodes which are able to communicate with others.

4.2.1. Mathematical Modelling of Link Failures

Nodes joining a DHT overlay usually randomly choose their own node identifier from a very large range of numbers. Also the output of hash functions can also be treated as a random number. The network seems to choose the node responsible for a specific key randomly. This property makes modelling the overlay relatively simple.

Fig. 3.
Ratio of successful lookups in the Kademlia overlay



The error ratio for the node with identifier *m* is given by (1):

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

where *n* is the total number of nodes ($0 \le m < n$). α sets the distribution of errors $\alpha=2$ for quadratic distribution. *c* is a constant setting the maximum number of errors. These parameters can be selected experimentally, and they depend on the actual size and properties of the underlying physical network.

Function (1) should output an integer number, as the error ratio multiplied by the number of all nodes, $n \cdot h(m)$, is an integer. For a high number of errors, the difference is negligible. Approximations using equation (1) will not be applicable to networks with a very low error rate, where $n^*h(m)$ is almost zero for the whole range of nodes. 0.3 errors have no real-world meaning, only 0 or 1 error.

As the underlying physical network, the Internet is not perfect; we also cannot expect the overlay to be so. Rather we can set a numeric expectation, for example we would like our network to be able to retrieve data in 99% of all cases. If the ratio of allowable errors is β= 1%, the probability of a successful lookup is 1-β, if the inequation $h(m) \le \beta$ holds for a given node. Those are the nodes, which can be accessed from most other ones.

As node identifiers are usually as large as 128 or 160 bits, the range of addresses can be treated as continuous. As all nodes bear randomly selected identifiers, and also the output of hash functions applied to keys seem to be random and evenly distributed over the range of possible identifiers, *m/n* is virtually a random number chosen from the interval [0,1). If we solve the inequality to express *m/n*, we get the number of nodes which match the specified criterion:

$$\frac{m}{n} \le \sqrt[\alpha]{\frac{\beta}{c}} \qquad (2)$$

Let *P'* be the probability of a successful lookup. As $0 \le m/n < 1$ holds, and *m/n* is randomly chosen over the interval, inequality (3) will hold for *P'*:

$$P' \le \sqrt[\alpha]{\frac{\beta}{c}} \qquad (3)$$

If the overlay employs replication, data is stored at *k* different points of the network. So we have *k* chance to choose different random numbers from the interval [0,1). If we manage to choose a suitable number at least once, the lookup will be successful. Calculating the probability of all lookups failing, and subtracting that value from 1, we get (4).

$$P = 1 - (1 - P')^{k} \qquad (4)$$

Formula (4) gives the probability of successful lookups with a given ratio of networks failures. The necessary level of replication can be calculated with the formula. *Fig. 4* shows the ratio of cases when the probability of successful lookups is at least 99% (*1%* failure allowed), as a function of network failures and level of replication.

As one can see, with a relatively high ratio of failing links (*10%*), replication *k=5* is enough to ensure successful lookups. If the overlay consists only of a small number, for example tens of nodes, *k=5* may seem too much. But this *k=5* can be used to any number of nodes. The formula gives results which closely match our simulation; the difference can only be seen for small error rates, as it was expected due to the approximation in (1).

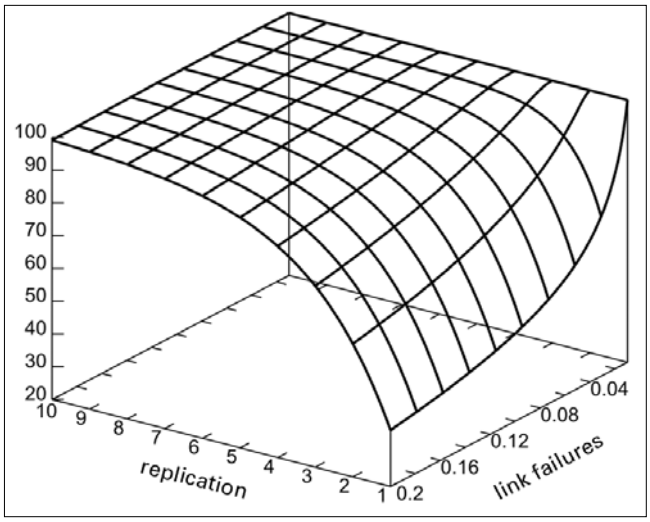


Fig. 4.
Approximated ratio of successful lookups in Kademlia

4.3. Broadcast messages in P2P overlays

Broadcast (one to all) messages in P2P overlays are not very common, due to the very big number of nodes. Usually no algorithm is designed to send these broadcast messages, as this contradicts with the main design goal of scalability. However there are applications which need this type of messaging, Komondor is an example. When a node has collected enough information making sure that an IP address belongs to an attacker, it initiates a broadcast message over the network. Another common application of broadcast messages is implementing lookups for partially given keywords, as this is not an elementary service in DHT networks (for example, one cannot lookup a partially given file name.)

The inherent topology of structured networks can be used to quickly and efficiently deliver broadcast messages. Using the built in topology will always give the best results. One reason for this, that the topology is built such a way that any node can be reached in logarithmically many steps, so the broadcast message will reach all nodes in logarithmic time.

The second is, that during sending the message, there will be no need to create new connections or initiate lookups. The topology can essentially be seen as an *implicit multicast tree*.

The Komondor system is an application where the fast sending of broadcast messages is essential. It is usually very easy to create reliable messaging over an unreliable channel, however detecting a packet loss needs quite long time. According to our tests, the broadcast algorithms presented in our article send the message in a few seconds to all nodes; to detect loss of a packet alone needs more time than this. If we do not try to resend the packets, the simulation will give us the shortest time in which the broadcast can be finished. Using replication, the time can be shorter than it is needed to detect packet loss. Simulating broadcast without resends will also give us the ratio of cases when the broadcast is successful, and is able to keep this time.

We developed three algorithms to send broadcast messages over Kademlia.

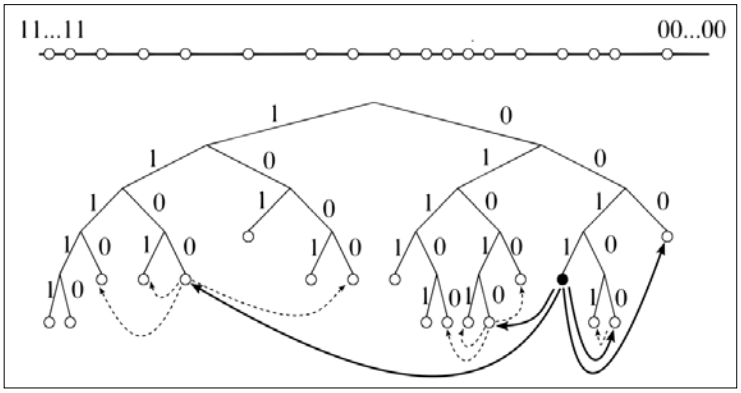
4.3.1. Broadcast Using Flooding

All nodes send received messages to any other nodes they know. As a specific message can be received in duplicates, every broadcast should be tagged with a unique identifier. 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, one can see the time the broadcast requires.

4.3.2. Broadcast Using the Topology

In the second algorithm, every subtree in the Kademlia overlay is assigned a node, which is responsible for broadcasting the message in its own tree *(Fig. 5)*. 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 (normal arrows). These nodes 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 shown using dashed lines. Broadcast using this method will be finished in logarithmic time.

Fig. 5. Broadcast messaging in Kademlia



Nodes forwarding messages must know which subtree they are responsible for. Every message is tagged with a small integer, which denotes the height of the subtree; this shows how many prefix bits the address of the subtree should share with the NodeID. The Kademlia protocol makes sure that at least one node is always known for every subtree; there is no need to maintain an auxiliary routing table for the broadcast.

Messages are forwarded to the subtree and all smaller trees:

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

This method is very cost-efficient as there are no duplicate messages. The number of messages sent grows exponentially, so the complete process takes logarithmic time. Problems can arise when there are packet losses on the network, as not only single nodes, but complete subtrees will miss the broadcast. Messages are actually directed to subtrees in this method: the original sender sends the message to the other 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.

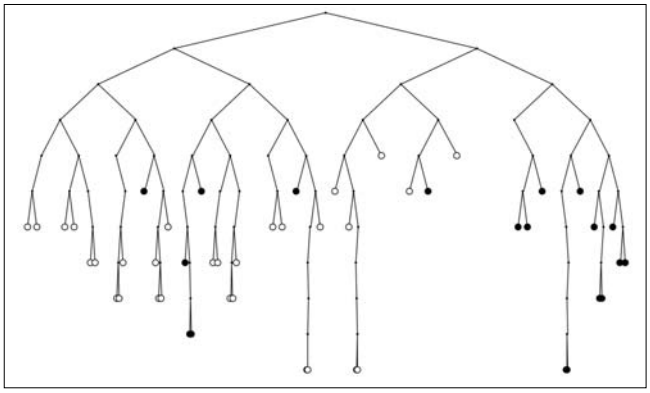


Fig. 6.
Implicit tree broadcast messaging in Kademlia

Fig. 6 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%, *independent* from 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.

4.3.3. 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 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 two to *k*, the size of k-buckets.

4.4. Comparison of broadcast algorithms

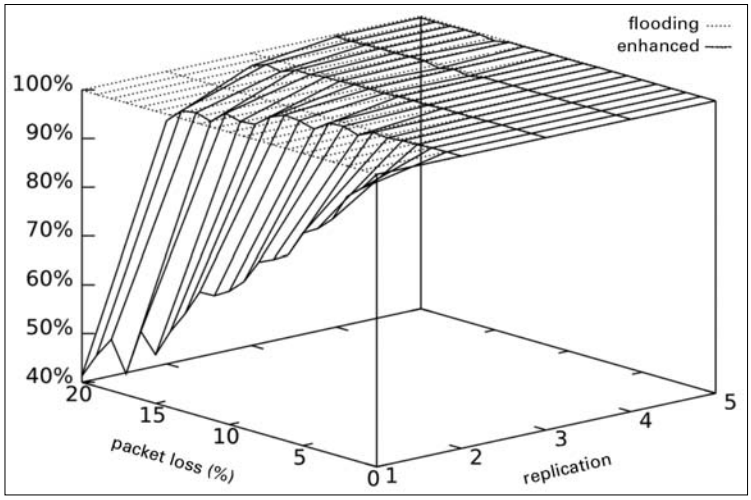
To evaluate the algorithms presented above, we developed a simple, application specific simulator. The program records the following data:

– number of all messages sent,
– number of messages per node,
– the number and ratio of nodes receiving the broadcast,
– the 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 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.

To evaluate the reliability of the algorithms, we simulated an overlay of 200 nodes. Packet loss ratio varied from 0% to 20%, replication from onefold to fivefold. Flooding almost always yields perfect results; the error is smaller than line width in *Fig. 7*. This is due to the enormous number of messages. The reliability of the enhanced algorithm is of course the same as the second for *k=1*, so it is not denoted individually. In turn, using *k=2*, this algorithm produces 90% reliability even if one fifth of the packets lost; *k=3* gives 97%.

Fig. 7.
Reliability of different broadcast algorithms in Kademlia



The time required to complete the broadcast is mainly determined by the latencies of the contacts stored in the k-buckets. If we go against the recommendation of the original Kademlia paper, and instead of the oldest contacts, we select a contact with low latency for the k-buckets, the time of lookups and broadcast both decreases significantly. The latency (or the round trip time, RTT) can easily be measured using PING messages, but it can also be approximated [6]. In the case simulated by our Kadsim application, the broadcast was two and a half times faster. Of course this ratio depends on the distribution of latencies, too.

The quickest method is of course the flooding *(Fig. 8)*, as messages sent in every possible direction will obviously travel the quickest way, too. Replication will speed up the algorithm for randomly selected nodes, and for RTT selected nodes, it will not. The implicit tree broadcast algorithm is the slowest, due to its rigidity. The third algorithm is between the previous two; using replication, it can be faster than the implicit tree algorithm using RTT selected nodes. This is also caused by messages travelling more than one way at a time.

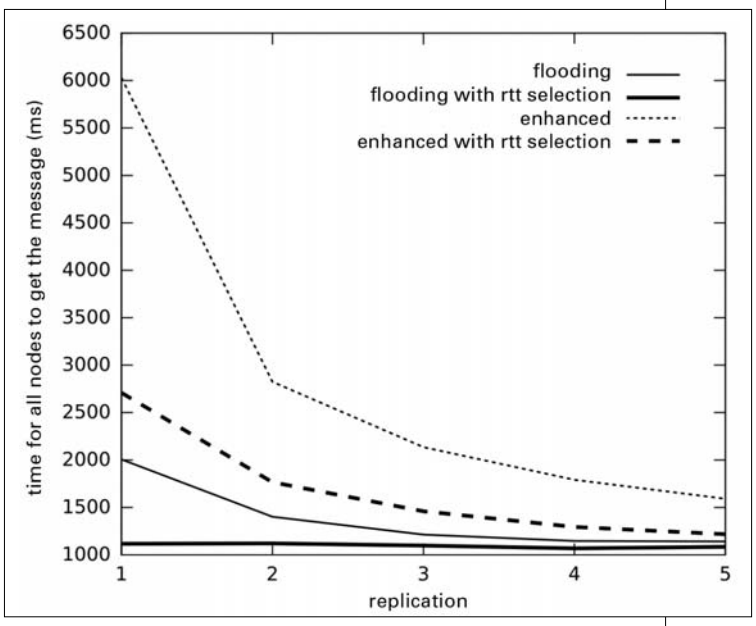


Fig. 8. The time needed for the broadcast

In the above figure, results from one hundred test runs were averaged. The lowest latency was around 15 ms, the average 0.5 s, and the highest was around 1.3 s.

5. Conclusions

The DHT-based intrusion prevention system presented in our paper is capable of creating a robust overlay of the participant software entities. Using a structured overlay network, the detection is distributed, but still it creates little processing and network traffic overhead. The reliability of the two elementary services, namely sending attack reports and broadcasting alerts can both be increased using replication. The only system-wide configuration parameter affecting the substrate, the level of this replication can be determined in advance, using the methods presented.

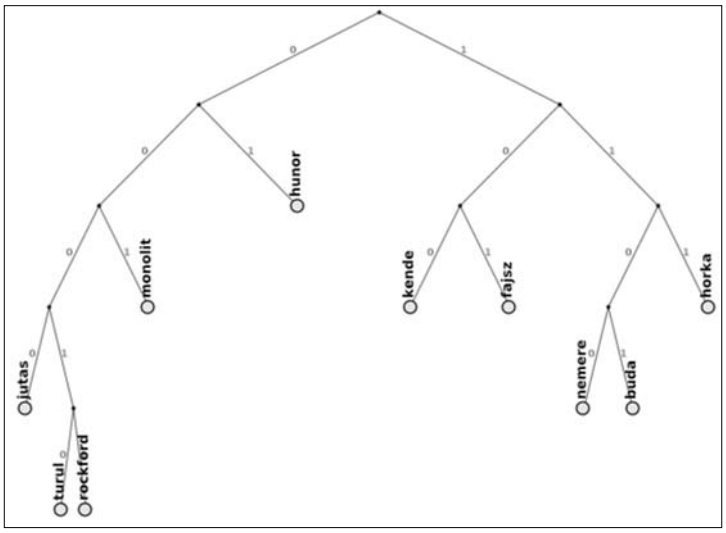


Fig. 9. A Komondor overlay in operation

Fig. 9 presents a screenshot of a smaller Komondor test overlay, with its binary tree topology. During its test runs lasting several months, it detected and prevented many intrusion attempts, and the substrate was proven to be stable. Attack reports which could be used by multiple nodes were alerts of SSH and HTTP intrusion attempts. The Snort application we used for intrusion detection logged many events which were not usable at all. For example computer viruses do not attack a single selected host for a long time. Against that kind of malware, the PROMIS system is more useful than ours [12].

The topic of our further research is the type of alerts to send on the overlay. One has to select the types of attacks, which are worth collecting and analyzing distributedly. Special attention must be paid to the case of Komondor nodes which run on different operating systems and applications: heterogeneity can increase security, as it is easier to see an attack manifesting if the system is immune. Using data collected this way may however be more difficult, as the protection must always be tailored to the host and environment in question.

Later research topics will include protection against malicious nodes building into the Komondor overlay itself. It is very easy to imagine that a compromised node sends alerts, attack reports about otherwise well-behaving clients; this way causing denial of service to the authenticated users of a system. This problem must be dealt with whatever distributed intrusion detection system one uses.

Authors

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 communications. In 2005, he won the second award at the Conference of Scientific Circle of Students, with his paper "Development of P2P Based Security Software". He published several technical papers and wrote chapters as co-author in the field of the collaborative security.

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. He published several technical papers, chapters and books. In 2001 he received the three-year Bolyai János Research Grant of the Hungarian Academy of Sciences. He lead a number of research projects. His main interests are internet-based media communications, multicasting, P2P communications, network intrusion detection systems, character encoding and VHDL-based system design.

References

[1] Czirkos Z.,
Developing a P2P Based Intrusion Detection System,
In Proc. of the Conf. of Scientific Circle of Students,
Budapest, 11-11-2005,
2nd Award (in Hungarian).

[2] Gnutella homepage,
http://www.gnutella.org/

[3] Snort – the de facto standard
for intrusion detection/prevention,
http://www.snort.org/
(Retrieved 26-11-2008)

[4] OSSEC – Open Source Host-based Intrusion
Detection System,
http://www.ossec.net/
(Retrieved 26-11-2008)

[5] P. Maymounkov and D. Mazieres,
Kademlia: A Peer-to-peer Information System Based
on the XOR Metric.
In Proc. of IPTPS02, Cambridge, USA, March 2002.
http://www.cs.rice.edu/Conferences/IPTPS02/

[6] F. Dabek, R. Cox, F. Kaashoek and R. Morris,
Vivaldi: A Decentralized Network Coordinate System.
In Proc. of the ACM SIGCOMM'04 Conference,
Portland, OR, August 2004.

[7] Z. Czirkos, G. Hosszú,
"On the Stability of Peer-to-Peer Networks
in Real-World Environments" – chapter in book,
Encycl. of Information Communication Technology,
2nd ed., Editors: Antonio Cartelli and Marco Palma,
Information Science Reference,
Hershey, USA, 2008. ISBN: 978-1-59904-651-8,
pp.622–630.

[8] D. Karger, E. Lehman, F. T. Leighton, M. Levine,
D. Lewin and R. Panigrahy,
Consistent hashing and random trees:
Distributed Caching Protocols for Relieving Hot Spots
on the World Wide Web.
In Proc. of the 29th Annual ACM Symposium on
Theory of Computing, May 1997.
pp.654–663.

[9] I. Stoica, R. Morris, D. Karger,
M. F. Kaashoek and H. Balakrishnan,
Chord: A Scalable Peer-to-peer Lookup Service for
Internet Applications.
Technical Report TR-819, MIT, March 2001.

[10] S. Rhea, D. Geels, T. Roscoe and J. Kubiatowicz,
Handling Churn in a DHT.
In Proc. of USENIX Technical Conf., June 2004.

[11] G. Hosszú, Z. Czirkos,
'Network-Based Intrusion Detection' chapter in book,
Encycl. of Internet Technologies and Applications,
Editors: Mário Freire and Manuela Pereira,
Information Science Reference,
Hershey, USA, 2007. ISBN: 978-1-59140-993-9,
pp.353–359.

[12] Vasileios Vlachos, Diomidis Spinellis,
A Proactive Malware Identification System based on
the Computer Hygiene Principles.
Information Management and Computer Security,
Vol. 15, No. 4, 2007.
pp.295–312.

[13] Feng Zhou, Li Zhuang, Ben Y. Zhao, Ling Huang,
Anthony D. Joseph and John Kubiatowicz,
Approximate Object Location and Spam Filtering
on Peer-to-peer Systems.
In ACM Middleware 2003.