S/Kademlia: A practicable approach towards secure key-based routing

Conference Paper · January 2008

DOI: 10.1109/ICPADS.2007.4447808 · Source: IEEE Xplore

2 authors:

Ingmar Baumgart
Karlsruhe Institute of Technology
37 PUBLICATIONS **797** CITATIONS
SEE PROFILE

Sergio Mies
University of São Paulo
148 PUBLICATIONS **1,884** CITATIONS
SEE PROFILE

S/Kademlia: A Practicable Approach Towards Secure Key-Based Routing

Ingmar Baumgart and Sebastian Mies
Institute of Telematics
Universität Karlsruhe (TH)
D–76128 Karlsruhe, Germany
Email: {baumgart, mies}@tm.uka.de

Abstract

Security is a common problem in completely decentralized peer-to-peer systems. Although several suggestions exist on how to create a secure key-based routing protocol, a practicable approach is still unattended. In this paper we introduce a secure key-based routing protocol based on Kademlia that has a high resilience against common attacks by using parallel lookups over multiple disjoint paths, limiting free nodeId generation with crypto puzzles and introducing a reliable sibling broadcast. The latter is needed to store data in a safe replicated way. We evaluate the security of our proposed extensions to the Kademlia protocol analytically and simulate the effects of multiple disjoint paths on lookup success under the influence of adversarial nodes.

1. Introduction

For a long time structured peer-to-peer networks like Chord [18] or Pastry [14] have only raised interest in the research community but were not largely used in real networks. Nowadays the situation in the Internet has changed significantly with the success of distributed filesharing applications. But also the emerging area of Massively Multiplayer Online Games has raised interest in using structured peer-to-peer networks to build scalable storage and communication services.

A major problem of completely decentralized peer-to-peer systems are security issues. Several common attacks against structured peer-to-peer networks have been identified [3, 16, 17]. Although many suggestions exist how to deal with such attacks in theory, there has been little work on building a practicable secure peer-to-peer network.

All widely deployed structured overlay networks used in the Internet today (i.e. BitTorrent, OverNet and eMule) are based on the Kademlia [11] protocol and are vulnerable to several attacks [5]. In this paper we analyze attacks on Kademlia networks and propose several practicable countermeasures.

The rest of this paper is organized as follows: In section 2 we provide some background on structured peer-to-peer networks and give an introduction to the Kademlia protocol. In section 3 we present an overview of common attacks against Kademlia. The detailed design of our security extensions to Kademlia is described in section 4. In section 5 we evaluate our design by simulation. Finally, section 6 covers related work and section 7 concludes.

2 Background

In this section we provide some background on structured peer-to-peer systems and describe some properties of the Kademlia protocol. A common service which is provided by all structured peer-to-peer networks is the *key-based routing layer (KBR)* [6]. This layer provides efficient routing to identifiers called *keys* from a large *identifier space* (typically n-bit integers modulo 2^n with $n = 128$ or 160). Every participating node in the overlay chooses a unique *nodeId* from the same id space and maintains a routing table with nodeIds and IP addresses of neighbors in the overlay topology. Depending on the overlay protocol the topology resembles a ring, hypercube or de Bruijn graph. Every node is responsible for a particular range of the identifier space, usually for all keys close to its *nodeId* in the id space. In this way KBR can be used to efficiently route a message to an arbitrary key by successively forwarding the message to overlay neighbors which have a *nodeId* closer to the destination key. This KBR layer can be used as building block for more complex tasks like a distributed storage service.

Although structured peer-to-peer systems like Chord, Pastry or Kademlia provide a similar KBR service, they differ in properties like routing table size and average lookup path length. In the following we focus on the Kademlia [11] protocol, because it is already insusceptible to several common attacks and at the same time simple to implement.

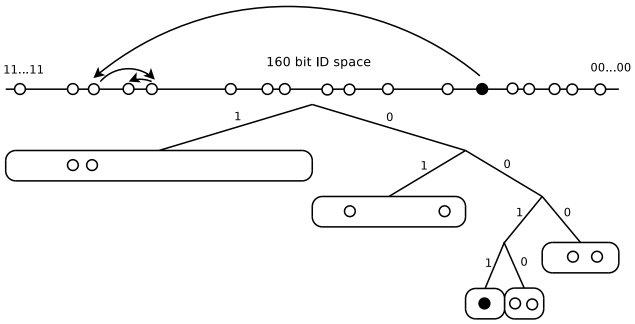

Figure 1. Routing table in Kademlia for a node with a nodeId prefix of 0011

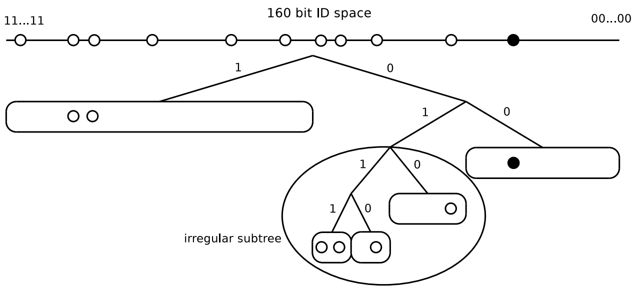

Figure 2. Kademlia routing table irregularities in a highly unbalanced tree

2.1. Kademlia

Kademlia [11] is a structured peer-to-peer system which has several advantages compared to protocols like Chord [18] as a results of using a novel XOR metric for distance between points in the identifier space. Because XOR is a symmetric operation, Kademlia nodes receive lookup queries from the same nodes which are also in their local routing tables. This is important in order that nodes can learn useful routing information from lookup queries and update their routing tables. In contrast Chord needs a dedicated stabilization protocol due to the asymmetric nature of the overlay topology. The XOR metric is also unidirectional: For any constant x there exists exactly one y which has a distance of $d(x, y)$. This ensures that lookups for the same key converge along the same path independently of the origination which is an important property for caching mechanisms.

In Kademlia every node chooses a random 160-bit *nodeId* and maintains a routing table consisting of up to 160 *k-buckets*. Every k-buckets contains at most *k* entries with <IP address, UDP port, NodeId> triples of other nodes. The parameter *k* is a redundancy factor to make the routing more robust by spanning several disjoint paths between overlay nodes. Buckets are arranged as a binary tree and nodes get assigned to buckets according to the shortest unique prefix of their *nodeId*s. An example routing table of a node with a *nodeId* prefix of 0011 is shown in figure 1. Initially, a node's routing table consists of a single bucket covering the entire *ID* space. When a node *U* learns of a new contact *C*, the node *U* inserts *C* in the appropriate bucket according to the prefix of *C's nodeId*. If this bucket already contains *k* nodes and the bucket's range includes *U's* own *nodeId*, the bucket is split into two new buckets. Otherwise the new contact is simply dropped.

A problem arises if nodeIds are unequally distributed. In this case the standard bucket splitting algorithm can lead to nodes not knowing their complete *k* neighborhood. An example for this is given in figure 2. This figure shows the routing table of a node *U* with *nodeId* prefix of 00. According to the splitting rule given above, there would be only one bucket for prefix 01 with *k* entries, which doesn't get split any futher. In this case the node with prefix 010, which is the closest node to node *U* could get dropped, leading to an incomplete neighborhood. To avoid this, the authors of Kademlia propose in these cases to also split buckets in which node *U's nodeId* isn't contained, resulting in an irregular subtree next to the bucket of *U's nodeId*. This exception to the bucket splitting algorithm makes the protocol more complex to implement, hence we propose an alternative approach in section 4.2.

3. Attacks on Kademlia

In this section we describe the taxonomy of possible attacks on Kademlia's key-based routing and data storage. In addition we present a theoretical estimation of lookup success rates in a Kademlia network with a fraction of adversarial nodes.

3.1. Attacks on the underlying network

We assume, that the underlying network layer doesn't provide any security properties to the overlay layer. Therefore an attacker could be able to overhear or modify arbitrary data packets. Furthermore we presume nodes can spoof IP addresses and there is no authentication of data packets in the underlay. Consequently, attacks on the underlay can lead to denial of service attacks on the overlay layer.

3.2. Attacks on overlay routing

Besides the attacks in the underlay there are many attacks which address the influence on overlay routing. This section summarizes the currently known attacks on overlay stability.

Eclipse attack: This attack tries to place adversarial nodes in the network in a way that one or more nodes are cut off from it, i.e. all messages are routed over at least one adversarial node. This gives the attacker the control over a part of the overlay network. Thus the Eclipse attack can "hide" some nodes from the overlay network. It can be prevented, first, if a node can not choose its *nodeId* freely and secondly, when it is hard to influence the other nodes routing table. Because Kademlia favors long-living nodes in its k-buckets and nodes are only added, if a bucket is not already full, the latter is easy to achieve as soon as the network has bootstrapped.

Sybil attack: In completely decentralized systems there is no instance that controls the quantity of nodeIds an attacker can obtain. Thus an attacker can join the network with lots of nodeIds until he controls a fraction m of all nodes in the network. Douceur [8] proved that this attack can not be prevented but only impeded. In centralized systems one might let nodes pay monetarily to join the network and bind this node to a particular natural person. In completely decentralized systems the only way a node can "pay" only with system resources (bandwidth, CPU power etc.) for authorization.

Churn attack: If the attacker owns some nodes he may induce high churn in the network until the network stabilization fails. Since a Kademlia node is advised to keep long-living contacts in its routing table, this attack does not have a great impact on the Kademlia overlay topology.

Adversarial routing: Since a node is simply removed from a routing table when it neither responds with routing information nor routes any packet, the only way of influencing the networks' routing is to return adversarial routing information. For example an adversarial node might just return other collaborating nodes which are closer to the queried key. This way an adversarial node routes a packet into its subnet of collaborators and neither the queried nor the closest node for a given key would be found. This can be prevented by using a lookup algorithm that considers multiple disjoint paths. The lookup succeeds when one path is free of adversarial nodes. Therefore, suppose m is the fraction of adversarial nodes, d the number of disjoint paths, (h_i) the path length distribution, then the probability that a lookup succeeds is given by:

$$P_K := \sum_{i=1}^{|(h_x)|} \left(h_i \cdot \left(1 - \left(1 - (1-m)^i \right)^d \right) \right)$$

This shows that particularly with regard to a moderate number of adversarial nodes, lookup algorithms can significantly benefit from disjoint paths as well as a low average path length.

3.3. Other attacks

Denial-of-Service: A adversarial may try to suborn a victim to consume all its resources, i.e. memory, bandwidth, computational power. Thus the protocol needs to have mechanisms to allocate resources in a secure way. Physical attacks, such as jamming or side-band attacks are not considered in this paper.

Attacks on data storage: Key-based routing protocols are commonly used as building blocks to realize a distributed hash table (DHT) for data storage. To make it more difficult for adversarial nodes to modify stored data items, the same data item is replicated on a number of neighboring nodes. Although attacks on data storage are not regarded in this paper, the key-based routing layer has to provide a secure neighborhood to a given key.

4. Design

In this section we propose a practicable secure Kademlia protocol. In section 4.1 we introduce a method to assign *nodeId*s in a secure way. Further a reliable sibling broadcast is proposed, which is needed to store replicated data in a secure way. Finally we explain how to secure the routing table maintenance.

4.1. Secure *nodeId* assignment

As known from section 3.2, it should be hard for an attacker to generate a large number of *nodeIds* (Sybil attack) nor choose the *nodeId* freely (Eclipse attack). Furthermore the *nodeId* should authenticate a node, i.e. no other node should be able to steal or fake the *nodeId*. The latter can be achieved with one of the two methods: Using a hash value over IP address and port or by hashing a public key. The first solution has a significant drawback because with dynamically allocated IP addresses the *nodeId* will change subsequently. It is also not suitable to limit the number of generated *nodeIds* if you want to support networks with NAT in which several nodes appear to have the same public IP address. Finally there is no way of ensuring integrity of exchanged messages with those kind of *nodeIds*. This is why we advocate to use the hash over a public key to generate the *nodeId*. With this public key it is possible to sign messages exchanged by nodes. Due to computational overhead we differentiate between two signature types:

- *Weak signature:* The weak signature does not sign the whole message. It is limited to IP address, port and a timestamp. The timestamp specifies how long the signature is valid. This prevents replay attacks if dynamic IP addresses are used. For synchronization issues the timestamp may be chosen in a very coarse-grained way. The weak signature is primarily used in `FIND_NODE` and `PING` messages where the integrity of the whole message is dispensable.

- *Strong signature:* The strong signature signs the full content of a message. This ensures integrity of the message and resilience against *Man-in-the-Middle* attacks. Replay attacks can be prevented with nonces inside the RPC messages.

Those two signature types can authenticate nodes and ensure integrity of messages. We now need to impede the Sybil and Eclipse attack. This is done by either using a crypto puzzle or a signature from a central certificate authority, so we need to combine the signature types above with one of the following:

- *Supervised signature:* If a signature's public key additionally is signed by a trustworthy certificate authority, this signature is called *supervised signature*. This signature is needed to impede a Sybil attack in the network's bootstrapping phase where only a few nodes exist in the network. A network size estimation can be used to decide if this signature is needed.

- *Crypto puzzle signature:* In the absence of a trustworthy authority we need to impede the Eclipse and Sybil attack with a crypto puzzle. In [3] the use of crypto puzzles for *nodeId* generation is rejected because they cannot be used to entirely prevent an attack. But in our opinion they are the most effective approach for distributed *nodeId* generation in an completely decentralised environment without trustworthy authority and should therefore be used to make an attack as hard as possible in such networks.

  For this reason we introduce two crypto puzzles as shown in figure 3. A static puzzle that impedes that the *nodeId* can be chosen freely and a dynamic puzzle that ensures that it is complex to generate a huge amount of *nodeIds*. H denotes a cryptographically secure hash function, $\oplus$ is the XOR operation, and the c_i denote crypto puzzle complexity. It is obvious that an increase of c_1 decreases the space of possible public keys therefore the size of the public key must be increased subsequently. c_1 is considered as a constant. Once the *nodeId* has been generated it stays fixed for a certain value. c_2 can be modified or extended over time when computational resources become cheaper.

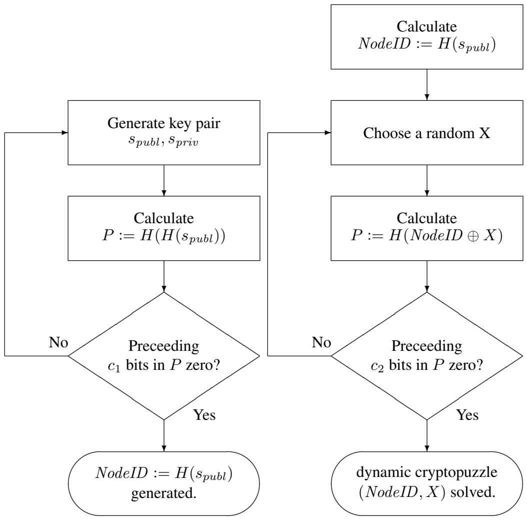

Figure 3. Static (left) and dynamic (right) crypto puzzles for nodeId generation

If a node receives a signed message it can now first validate its signature and then check if the crypto puzzles were solved. Both operations have $O(1)$ complexity for a constant public key size while crypto puzzle creation has $O(2^{c_1} + 2^{c_2})$ complexity.

4.2. Reliable sibling broadcast

Siblings are nodes which are responsible for a certain key-value pair that needs to be stored in a DHT. In the case of Kademlia those key-value pairs are replicated over the k closest nodes (we remember: k is the bucket size). In this paper we want to consider this number of nodes independently from the bucket size k and introduce the number of siblings as a parameter s. This makes sense because the bucket size usually defines the redundancy of overlay connectivity and not the number of replicas that need to be stored by the DHT.

A common security problem is the reliability of sibling information which arises when replicated information needs to be stored in the DHT which uses a majority decision to compensate for adversarial nodes. Since Kademlia's original protocol converges to a list of siblings, it is complicated to analyze and prove the coherency of sibling information. For this reason we introduce a sibling list of size $\eta \cdot s$ per node, which ensures that each node knows at least s siblings to a *ID* within the nodes' siblings range with high probability. We now have to determine η and prove that the constraints hold with high probability. This has already been done by Gai and Viennot in their paper about the Broose DHT [10] where a brotherhood list is constructed in the same way. Since this proof unnecessarily uses Chernoff

	$c = 1.5$	$c = 2.0$	$c = 2.5$	$c = 3.0$
$s = 8$	$0.8950 \cdot 10^{-1}$	$1.0000 \cdot 10^{-2}$	$0.7786 \cdot 10^{-3}$	$0.4750 \cdot 10^{-4}$
$s = 16$	$0.3440 \cdot 10^{-1}$	$0.6600 \cdot 10^{-3}$	$0.5464 \cdot 10^{-5}$	$0.2590 \cdot 10^{-7}$
$s = 20$	$0.2187 \cdot 10^{-1}$	$0.1763 \cdot 10^{-3}$	$0.4791 \cdot 10^{-6}$	$0.6352 \cdot 10^{-9}$
$s = 32$	$0.5925 \cdot 10^{-2}$	$0.3617 \cdot 10^{-5}$	$0.3506 \cdot 10^{-9}$	$0.1022 \cdot 10^{-13}$

Table 1. Probability for an incomplete sibling list

bounds we review the proof here:

As mentioned before the XOR metric is unidirectional, which means that for a fixed x and for each $d_{\oplus}(x, y)$ exactly one y exists. Consider two *nodeId*s chosen at random, then the probability that one *nodeId* y is smaller than *nodeId* x is given by $\frac{x}{2^n}$ (n is the number of bits of the *nodeId*). Let N be the number of nodes in the network, then the *average* distance with the XOR metric between two adjacent nodes is $\frac{2^n}{N}$. Now consider the distance

$$d_N(\mu) = \mu \cdot \frac{2^n}{N}$$

of two nodes with *nodeId* x and y, then the expected number of nodes $N(x, y)$ between x and y is $E[N(x, y)] = \mu$ and the probability that a node is placed between x and y is given by $\frac{\mu}{N}$. Since *nodeId*s are randomly chosen the actual number of nodes between x and y varies from the expected value μ. Therefore the probability that there are less than s nodes between x and y at distance $d_N(cs)$ is given by:

$$\Pr[N(x, y) < s] = \sum_{i=0}^{s-1} \binom{N}{i} \left(\frac{cs}{N}\right)^i \left(1 - \frac{cs}{N}\right)^{(N-i)}$$

This probability is computeable for small $s > 0$, since the following is imperative:

$$\binom{n}{k} = \prod_{i=1}^{k} \frac{n + 1 - i}{i}$$

So the use of Chernoff bounds are not necessary. This makes this solution more accurate and the following probabilities depending on c and s with an upper bound network size of $N = 10^{10}$ can be computed as shown in Table 1.

The remainder of the proof follows [10]. The interesting fact is that it is shown that a value $\eta \geq (2 \cdot c) \geq 5$ is sufficient to satisfy our needed constraints w.h.p.

Thus in S/Kademlia the routing table consists of a list of n k-buckets holding nodes with a distance d with $2^{i-1} \leq d < 2^i, 0 \leq i \leq n$ and a sorted list of siblings of size $\eta \cdot s$. The special subtree handling introduced in section 2.1 can be omitted because of the newly introduced sibling list. Because routing tables in Kademlia are implicitly refreshed by incoming lookup requests and many of the nodes in our sibling table would otherwise have to be stored in Kademlia's k-buckets, the additional communication overhead for maintaining the sibling table is low.

4.3. Routing table maintenance

Kademlia uses a reactive approach to maintain routing tables. Since the XOR metric ensures that all iterative lookups converge along the same path, Kademlia can learn about the existence of new nodes from incoming RPCs. To secure routing table maintenance in S/Kademlia we categorize signaling messages to the following classes: Incoming signed RPC requests, responses or unsigned messages. Each of those messages contains the sender address. If the message is weakly or strong signed, this address can not be forged or associated with another *nodeId* (see section 4.1). We call the sender address *valid* if the message is signed and *actively valid*, if the sender address is valid and comes from a RPC response. Kademlia uses those sender addresses to maintain their routing tables.

Actively valid sender addresses are immediately added to their corresponding bucket, when it is not full. Valid sender addresses are only added to a bucket if the *nodeId* prefix differs in an appropriate amount of bits χ (for example $\chi > 32$). This is necessary because an attacker can easily generate *nodeIds* that share a prefix with the victims *nodeId* and flood his buckets, because buckets close to the own *nodeId* are only sparsely filled in Kademlia. Sender addresses that came from unsigned messages will be ignored. If a message contains more information about other nodes, then each of them can be added by invoking a ping RPC on them. If a node already exists in the routing table it is moved at the tail of the bucket.

4.4. Lookup over disjoint paths

In section 3.2 we have shown the importance of using multiple disjoint paths to lookup keys in a network with adversarial nodes. The original Kademlia lookup iteratively queries α nodes with a `FIND_NODE` RPC for the closest k nodes to the destination key. α is a system-wide redundancy parameter such as 2. In each step the returned nodes from previous RPCs are merged into a sorted list from which the next α nodes are picked. A major drawback of this approach is, that the lookup fails as soon as a single adversarial node is queried.

We extended this algorithm to use d disjoint paths and thus increase the lookup success ratio in a network with adversarial nodes. The initiator starts a lookup by taking the k closest nodes to the destination key from his local routing table and distributes them into d independent lookup buckets. From there on the node continues with d parallel lookups similar to the traditional Kademlia lookup. The lookups are independent, except the important fact, that each node is only used once during the whole lookup process to ensure that the resulting paths are really disjoint. By using the sibling list from section 4.2 the lookup doesn't

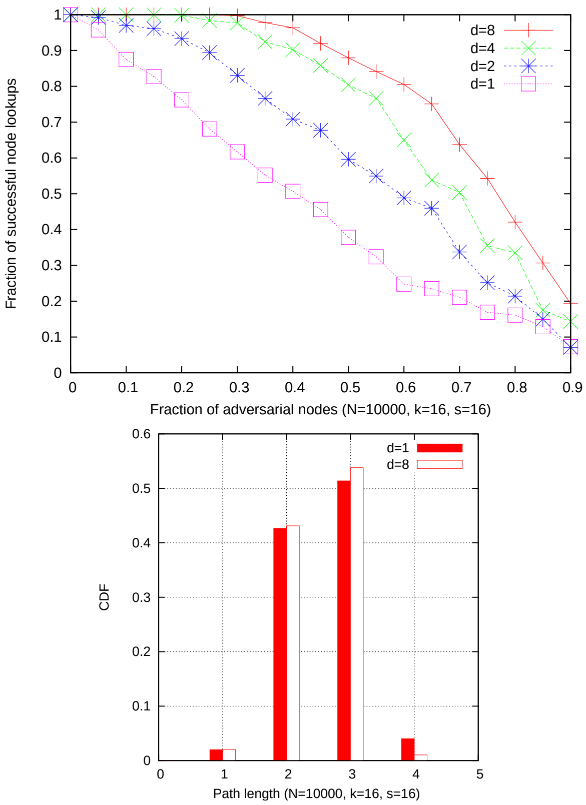

Figure 4. Fraction of successful node lookups with a fixed bucket size $k = 16$

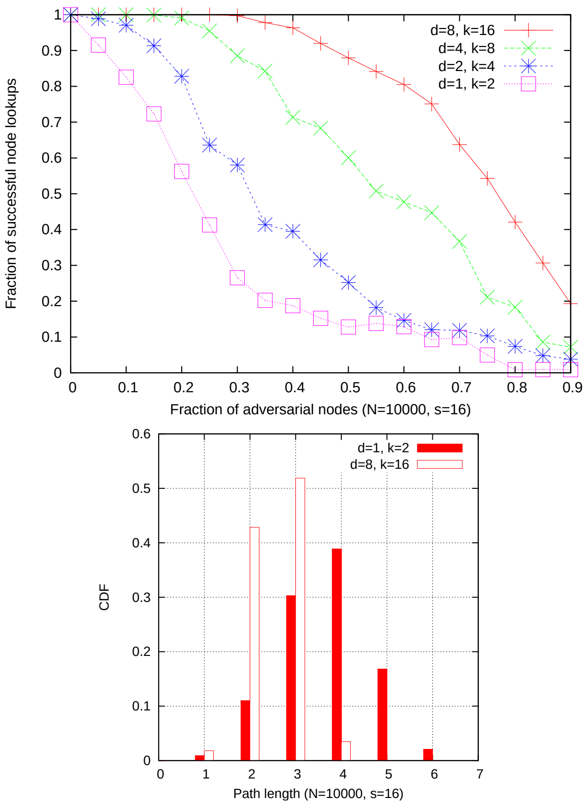

Figure 5. Fraction of successful node lookups with an adaptive bucket size $k = 2d$

converge at a single node but terminates on d close-by neighbors, which all know the complete s siblings for the destination key. Hence a lookup is still successful even if $k - 1$ of the neighbors are adversarial.

5. Evaluation

In this section we evaluate S/Kademlia with OverSim [2], a flexible framework for overlay simulation. We describe the simulation setup and procedure and finally the results of our simulations.

5.1. Simulation assumptions

We simulate adversarial nodes with the following assumptions: An adversarial node returns data that compromises the network in a worst case scenario. So in the case of a `FIND_NODE` RPC the worst behaviour would only return other collaborating nodes which are closer to the target *nodeId*. The adversarial also harvest other existing valid *nodeId*s in order to map them to a false transport addresses. This is the worst case since other reactions like an empty result or invalid data can be detected and the node would be removed from the network, i.e. the node would be considered as a stale contact.

On the other hand we assume that well behaving nodes return adversarial node information in an equally distributed manner. This assumption is confirmed in section 4 because it is nearly impossible for an adversarial node to influence other nodes' routing tables.

With these assumptions we expect that a lookup of a node or siblings is successful if no adversarial node is on one path to the responsible node. In the case of a parallel lookup on multiple paths we simply stop pursuing each path that hit an adversarial node and the lookup is only considered successful, if one path is free of any adversarial nodes.

5.2. Simulation procedure

To keep the simulation efficient we first create a static Kademlia overlay network with N nodes that is fully stabilized. Then we continue by processing N node lookups and evaluate the fraction of successful queries. This process is repeated with increasing adversarial nodes by 5% until 90% of the nodes are adversarial. Since we do not evaluate churn all simulations are done with a parameter of $\alpha = 1$ and we assume that the network stays in a stable state after the bootstrapping phase, but under the influence of adversarial nodes.

5.3. Results

We simulated two setups with a network size of $N = 10000$ nodes, $s = 16$ siblings using $d \in \{1, 2, 4, 8\}$ disjoint paths. In Figure 4 we show the fraction of successful lookups dependant on the number of adversarial nodes for a fixed bucket size of $k = 16$. For $d = 1$ the lookup process is similar to a standard Kademlia lookup. The figure clearly shows that by increasing the number of parallel disjoint paths d the fraction of successful lookups can be considerably improved. In this case the communication overhead increases linearly with d. We also see that with $k = 16$ there is enough redundancy in the k-buckets to actually create d disjoint paths.

In the second setup we adapted $k = 2 \cdot d$ to the number of disjoint paths to keep a minimum of redundancy in the routing tables and consequently reduce communication overhead. The results in figure 5 show, that a smaller k leads to a smaller fraction of successful lookups compared to figure 4. The reason for this is the increased average path length due to the smaller routing table as shown in the path length distribution diagram.

We conclude that $d = 4..8$ with a $k = 8..16$ is a good choice for S/Kademlia. Higher values of d and k seem not worth the additional communication costs. Larger values for k would also increase the probability that a large fraction of buckets are not full for a long time. This unnecessarily makes the routing table more vulnerable to Eclipse attacks.

Since we present simulations with $N = 10000$ nodes only, one might argue that this is a rather small number of nodes and not comparable to huge networks. In fact the path length highly corelates with the fraction of successful lookups. On the other hand the network topology can be easily tuned to have a smaller diameter and therefore a shorter average path length. This is usually done by considering multiple bits b of the *nodeId* in each step. So the network can be tuned to the level of security needed in different scenarios.

6. Related Work

Castro et al. [3] study attacks on routing of messages in structured peer-to-peer overlays. They propose several defenses to secure the join process, routing table maintenance and message forwarding. The secure assignment of nodeIds is delegated to a central trusted certification authority.

Sit and Morris [16] present a categorization of attacks against peer-to-peer distributed hash tables on the basis of Chord, CAN and Pastry. They state that an important step to defend these attacks is detection by defining verifiable system invariants. For example nodes can detect incorrect lookup routing by verifying that the lookup gets "closer" to the destination key.

Srivatsa and Liu [17] investigate three security threats in DHT-based P2P systems. First they present an attack on the routing scheme, in which a single adversarial node can block all lookup requests in the absence of alternate paths. Therefore they highlight the importance of several alternate optimal paths in conjunction with the feasibility to detect incorrect lookup results. Furthermore they present an attack on the data placement scheme and show that replication alone is not sufficient to tolerate attacks by adversarial nodes, but has to be combined with cryptographic techniques to be effective. Finally they show that the *nodeId* selection process has to be restricted to prevent adversarial nodes from corrupting specific data items.

Awerbuch and Scheideler [1] present a theoretical DHT which is provably robust against adversarial join-leave as well as insert-lookup attacks. The design of the DHT is high level and it is an open question how hard it would be to transform their ideas into a practicable protocol. Another DHT which can provably deal with join-leave attacks is S-Chord [9], though it is limited to a linear number of adversarial join requests.

Cerri et al. [4] focus on attacks that arise from the unlimited choice of nodeIds and exemplify their findings with the Kademlia protocol. They propose to limit free *nodeId* selection by coupling IP address and port to the *nodeId* by a hash function. To make it harder for adversarial nodes to attack specific data items, they propose that data items should be stored at a temporary key, which is regularly rotated. This is done by hashing the data item's key with some temporal information to compute the temporary key,

There are several papers in which countermeasures against Sybil attacks are proposed: In [13] Rowaihy et al. present an admission control system for structured peer-to-peer systems. The systems constructs a tree-like hierarchy of cooperative admission control nodes, from which a joining node has to gain admission. Another approach [7] to limit Sybil attacks is to store the IP addresses of participating nodes in a secure DHT. In this way the number of nodeIds per IP address can be limited by querying the DHT if a new node wants to join.

Singh et al. [15] study the impacts of Eclipse attacks on structured overlay networks and propose to defend against this attack by letting nodes audit each others connectivity. The idea is, that a node mounting an Eclipse attack has a node degree higher than average.

Nielson et al. [12] regard the class of rational attacks. They assume that a large fraction of nodes in a peer-to-peer system are selfish and try to maximize their consumption of system ressources while minimizing the use of their own.

7. Conclusion

In this paper we presented a secure key-based routing protocol based on Kademlia. Although the elegant routing table maintenance makes Kademlia already insusceptible to some attacks, we have shown that there are several vulnerability that make it easy for adversarial nodes to gain control of the network.

We propose several practicable solutions to make Kademlia more resilient. First we suggest to limit free nodeId generation by using crypto puzzles in combination with public key cryptography. Furthermore we extend the Kademlia routing table by a sibling list. This reduces the complexity of the bucket splitting algorithm and allows a DHT to store data in a safe replicated way. Finally we propose a lookup algorithm which uses multiple disjoint paths to increase the lookup success ratio.

The evaluation of S/Kademlia in the simulation framework OverSim has shown, that even with 20% of adversarial nodes still 99% of all lookups are successful if disjoint paths are used. We believe that the proposed extensions to the Kademlia protocol are practical and could be used to easily secure existing Kademlia networks.

Acknowledgment

This research was supported by the German Federal Ministry of Education and Research as part of the *ScaleNet* project 01BU567 and by the BW-FIT support program as part of the *SpoVNet* project.

References

[1] B. Awerbuch and C. Scheideler. Towards a scalable and robust dht. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 318–327, New York, NY, USA, 2006. ACM Press.

[2] I. Baumgart, B. Heep, and S. Krause. OverSim: A flexible overlay network simulation framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, May 2007.

[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, 2002.

[4] D. Cerri, A. Ghioni, S. Paraboschi, and S. Tiraboschi. Id mapping attacks in p2p networks. In *Global Telecommunications Conference, GLOBECOM'05. IEEE*, 2005.

[5] S. A. Crosby and D. S. Wallach. An analysis of two bittorrent distributed trackers. Presentation at the South Central Information Security Symposium (SCISS '06), 2006.

[6] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, volume Volume 2735/2003, pages 33–44, 2003.

[7] J. Dinger and H. Hartenstein. Defending the sybil attack in p2p networks: Taxonomy, challenges, and a proposal for self-registration. *ares*, 0:756–763, 2006.

[8] J. R. Douceur. The sybil attack. In *IPTPS '02: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[9] A. Fiat, J. Saia, and M. Young. Making chord robust to byzantine attacks. In *ESA*, Lecture Notes in Computer Science, pages 803–814. Springer, 2005.

[10] A.-T. Gai and L. Viennot. Broose: a practical distributed hashtable based on the de-bruijn topology. In *Fourth International Conference on Peer-to-Peer Computing, 2004*, pages 167–174, aug 2004.

[11] P. Maymounkov and D. Mazires. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7-8, 2002. Revised Papers*, volume Volume 2429/2002, pages 53–65, 2002.

[12] S. Nielson, S. Crosby, and D. Wallach. A taxonomy of rational attacks. In *4th International Workshop on Peer-To-Peer Systems*, Ithaca, New York, USA, February 2005.

[13] H. Rowaihy, W. Enck, P. Mcdaniel, and T. La-Porta. Limiting sybil attacks in structured peer-to-peer networks. Technical Report NAS-TR-0017-2005, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania StateUniversity, University Park, PA, USA, 2005.

[14] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001 : IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001. Proceedings*, volume Volume 2218/2001, pages 329+, 2001.

[15] A. Singh, T.-W. J. Ngan, P. Druschel, and D. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *In Proceedings of INFOCOM 06, Barcelona, Spain. April 2006*, 2006.

[16] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *IPTPS '02: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 261–269, London, UK, 2002. Springer-Verlag.

[17] M. Srivatsa and L. Liu. Vulnerabilities and security threats in structured overlay networks: A quantitative analysis. In *ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 252–261, Washington, DC, USA, 2004. IEEE Computer Society.

[18] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, feb 2003.