Purpose and security analysis of RASTER

Oliver Gasser
Advisor: Christian Grothoff
Seminar Future Internet SS2010
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: gasser@in.tum.de

ABSTRACT

In this paper we survey the purpose of the RASTER routing protocol and study its input on a P2P network's security. Routing in P2P-systems using adaptive distributed hash tables (DHT) poses different challenges than routing in traditional deterministic DHT-based P2P systems. Routing algorithms designed for deterministic DHTs do not always find those shortest paths in adaptive networks. The routing protocol RASTER tries to solve this problem. It minimizes the storing and forwarding overhead and has a good routing performance. We study RASTER(a)'s performance and examine the impact of faulty or malicious nodes on a P2P system which uses the RASTER routing protocol and discover that such a system is more vulnerable to routing manipulation attacks or erroneous nodes.

Keywords

Distributed hash table, Security, Adaptive DHT, Randomized DHT, Routing

1. INTRODUCTION

1.1 Terminology

Participants in a Peer-to-Peer (P2P) network are called nodes. A hash table matches a key k to its value v. A distributed hash table (DHT) is a hash table distributed over multiple nodes. A routing protocol for DHT tries to retrieve the value v for a certain key k as fast and efficiently as possible. If a certain v is not found at the node itself the node has to make a decision to which neighbor it forwards the query.

In a traditional *deterministic* DHT [6, 7, 9] each node selects its neighbors deterministically. This means, that a node has no influence over the nodes which will be its neighbors. Each node selects a certain fixed number of neighbors, the degree k of the overlay structure. Neighbors are selected based on their position in the overlay's hash space. A drawback of this procedure is that it does not consider the heterogeneity of nodes, i.e. the differences in nodes' Internet connectivity, service capacity, lifetime in the system or willingness to perform a certain task.

Adaptive DHT P2P systems overcome this shortcomings by giving nodes some flexibility to select their neighbors. Specifically a node chooses its neighbors based on characteristics such as their Internet connectivity, service capacity, lifetime in the system and service willingness. The nodes are, however, not chosen totally at random as the term *randomized* DHT may suggest in Wang et al.'s paper [11]. Total randomness in neighbor selection would lead to some problems which will be discussed in Section 3.1. Each node always has some near neighbors in order to ensure that a request can be routed towards its destination without looping back and forth. It has been shown [4] that picking neighbors with certain randomness leads to overall better performance specifically in terms of path latency, resilience to churn and local convergence.

1.2 Routing in adaptive DHTs

Unfortunately greedy routing, the dominant routing strategy in deterministic DHTs, falls short of finding the shortest overlay paths in adaptive DHTs. This strategy forwards queries to neighbors who are thought to be closest to the destination node, but do not respect the DHT's nondeterministic overlay structure. The Neighbors of Neighbors (NoN) strategy proposed by Manku et al. [5] performs better than greedy routing but is still far from optimal. In order to get better routing performance, NoN routing could be generalized, i.e. nodes also save 3rd, 4th etc. degree neighbors in their routing tables. However, this creates a huge space and communication overhead.

RASTER [11], a routing protocol for adaptive P2P networks, addresses this problem. It solves the overhead problem caused by having an exponentially increasing number of neighbors in routing tables, by aggregating routing states and encoding routing tables as *bitmaps*. It maintains routing information within a radius of a. It outperforms NoN and is more resilient to churn (nodes joining and leaving the network). RASTER guarantees the discovery of shortest-path overlay routes if the destination d is within a hops, but not otherwise.

In this paper we are describing RASTER in more detail. We then present measurement data investigating the claim of RASTER(a)'s better performance and survey the performance impact of changing parameters. Additionally we are also examining the impact of more information on the neighboring topology on a P2P network's security. If a malicious node joins a network which uses the RASTER algorithm, it has a greater influence factor than by joining a network relying solely on greedy routing.

The rest of this paper is structured as follows. In Section 2 we will examine Chord [9] as an example of a routing algorithm for deterministic DHTs and then in Section 3 we will

take a look at general routing strategies and see why greedy routing encounters problems in adaptive DHTs. Thereafter, as a possible solution for these problems, the bitmap concept and the RASTER routing algorithm [11] are introduced in Section 4. In Section 5 we do a performance comparison of RASTER's two different forwarding decision implementations. In Section 6 we discuss the impact of faulty or malicious nodes and we conclude this paper in Section 7 with the result of our inquiries.

2. DETERMINISTIC DHT ROUTING

This section will describe Chord [9], a typical, simple, deterministic DHT routing algorithm.

In Chord, a hash function assigns each node an m-bit ID by hashing the node's IP address. IDs are ordered in an *identifier circle* modulo 2^m. The key k is assigned to a node as follows: (1) If a node n exists with the same ID as the key k, then k is assigned to n, or (2) if there is no such node, then k is assigned to the successor node or $successor(k)$ which is the first actual node on the circle following k's ID.

For key location purposes each node n maintains a *finger table*, a routing table with m (number of bits of the hash space) entries. Each entry in this table consists of a Chord ID and an IP address. The i^{th} entry in the table points to the node $s = successor(n+2^{i-1})$ for $1 \leq i \leq m$. This is the first node which succeeds n by at least 2^{i-1}. Note that these neighbors in the finger table are selected deterministically.

In our example the nodes' IDs have a length of 6 bit ($m = 6$) and there are 10 nodes in the system. Figure 1 illustrates the finger table entries of node 48.

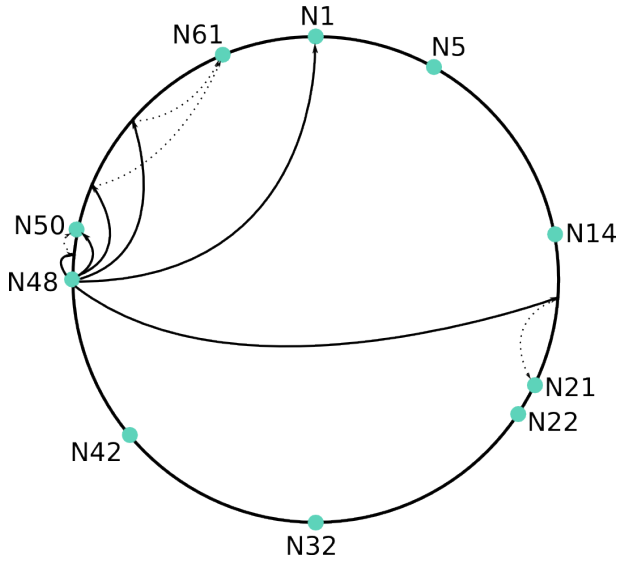

Figure 1: Chord neighbors of node with ID 48.

If a node n gets a request to retrieve the value v for the corresponding key k it first checks if itself is responsible for this key, i.e. $k \in (n, successor(n))$. If that is the case v is either stored at n and returned right away or can not be found in the DHT. Otherwise n forwards the request to the neighbor with the largest ID m smaller or equal to k. More formal:

$$\max_{m \in FT} \{m \leq k\}$$

where FT is the set of finger table IDs. This algorithm is of complexity $\mathcal{O}(\log n)$ as each node has $\log n$ neighbors and after at most $\mathcal{O}(\log n)$ hops the node responsible for the value v has been found. Each forwarding decision at least halves the distance to the destination.

3. ROUTING STRATEGIES IN DHTS

3.1 Greedy routing

Greedy routing is a strategy which bases its forwarding decision as follows: A node n forwards a request to that neighbor W^*, which is closest to the destination d. In a more formal way it selects W^* of its neighbors $W_1, W_2, \ldots, W_k$ for which $|d - W^*|$ is minimal. Assuming that the out degree of each node is $\mathcal{O}(\log n)$, the storage and forwarding overhead for each node in a deterministic DHT is $\mathcal{O}(\log n)$, the average path length between two arbitrary nodes is $\mathcal{O}(\log n)$.

As shown in Section 2, nodes in deterministic DHT networks select their neighbors based on a specific algorithm. For demonstration purposes we assume that the nodes are aligned like a matrix in the hash space and two nodes are neighbors, if they are side by side in a matrix' row or column. In an environment with deterministic neighbor selection, the greedy route always leads to shortest overlay paths (see Figure 2 (a)). Algorithms that are based on the greedy routing strategy do not find shortest paths in adaptive DHTs, i.e. with a somewhat irregular overlay structure. By forwarding the message to a neighbor they come closer to the destination. However, it is not guaranteed that this really is the route with the fewest hops. It is possible that the neighbor, to whom the request has been forwarded to, has neighbors which come only a little closer to the destination node d (see Figure 2 (b)).[1] The origin of the problem is that although you make a locally optimal choice, i.e. you make the biggest step towards the destination, you can't assume the same about it globally as in adaptive networks you don't have sufficient knowledge about the structure of the node's neighbors, the neighbors' neighbors etc.

Greedy routing (e.g. Chord), however, is not optimal for adaptive networks, i.e. where each node chooses its neighbors not deterministically but according to the neighbor's latency, service capacity, lifetime in the DHT etc. To find a better solution than greedy we need to survey the different requirements for routing in deterministic and adaptive DHTs.

3.2 NoN routing

An approach similar to greedy but specifically designed for adaptive DHTs is *Neighbor of Neighbor routing* (NoN) proposed by Manku et al. in [5]. In this algorithm each node not only saves information about its neighbors but also of its neighbors' neighbors (also called neighbors of 2^{nd} degree). When making a decision to which node to forward a request, the node forwards it to neighbor W^*, which has itself

[1] If the nodes chose neighbors at random as the term randomized DHT in [11] suggests, it would even be possible that there is no neighbor which gets locally closer to d, resulting in the wrong peer being identified as the target.

a neighbor closest to the destination node d. I.e. n identifies W^*, which can be of n's 1^{st} and 2^{nd} degree neighbors, and forwards the request to W^*. The average path length between two arbitrary nodes with NoN routing in a DHT where each node has $\mathcal{O}(\log n)$ neighbors is $\mathcal{O}(\frac{\log n}{\log \log n})$. This performance improvement compared to greedy routing comes with a significant drawback: Each node's routing tables contain $\mathcal{O}(\log^2 n)$ entries and the communication overhead is also greater.

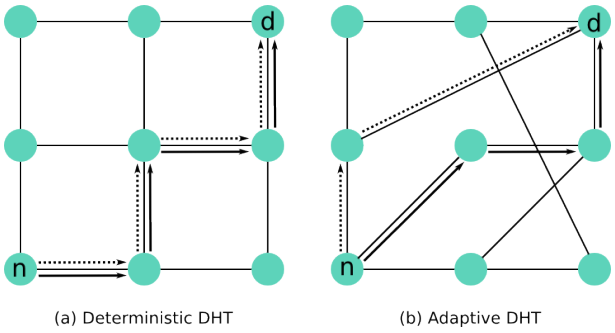

Figure 2: Difference between shortest path route (dashed line) and greedy route (solid line).

The NoN strategy performs slightly better than greedy routing as it can avoid neighbors which are perceived to be close to the destination node d but where then their neighbors are not rapidly progressing towards d. NoN is not the panacea as it only includes neighbors of 1^{st} and 2^{nd} degree. As pointed out in [11] (1) the average number of overlay hops independent of the overlay structure approaches the Moore bound $\lceil \log_k n \rceil$, where n is the number of nodes and k the degree. (2) The performance gap between shortest route and chosen route can be bridged by generalizing NoN routing, i.e. nodes also maintain information about higher degree neighbors. This, however, is not practical as it comes with high communication and storage overhead. The RASTER routing protocol, which will be explained in detail in the next section, addresses this issue.

4. RASTER ROUTING PROTOCOL

This section describes the definition and reasoning behind using bitmaps instead of traditional routing tables, construction of and operations on such bitmaps and the RASTER decision procedure in detail.

4.1 Bitmaps

A *bitmap* is an encoded representation of a set of nodes in a DHT network. It is important to note that the size of the bitmap is independent of the number of nodes in the DHT.

DEFINITION 1. *$B_s^{a,r}$ is a bitmap of resolution r, from an arbitrary node s and covering a radius a. It is an r-bit binary string representing the overlay positions that can be reached from node s within a hops. The system's hash space is to be partitioned evenly into r bins of the same size. If the i^{th} bit $(0 \leq i \leq r-1)$ in $B_s^{a,r}$ is set to 1, at least one node in bin i is reachable within a hops from node s.*

$B_s^{1,r}$ therefore represents the overlay neighbors of node s, $B_s^{2,r}$ neighbors of 2^{nd} degree and so on. See Figure 3 and

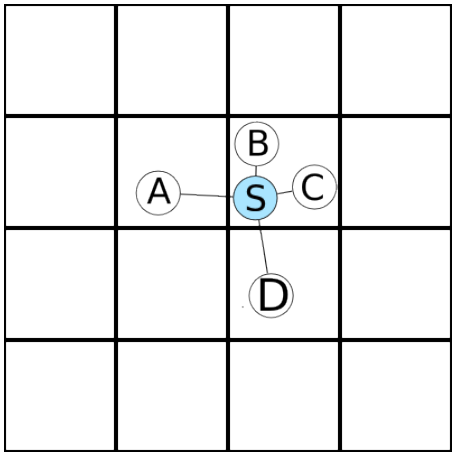

Figure 3: Constructing $B_S^{1,16}$ = 0000 0110 0010 0000.

Figure 4 on how to construct bitmaps. Contrary to traditional routing tables these bitmaps hide routing details, which are not relevant for forwarding decisions. Bitmaps don't contain entries for every single node but combine them into bins. Furthermore a node only retains IP addresses for its direct neighbors and not for all routing table entries. This makes bitmaps much smaller than routing tables.

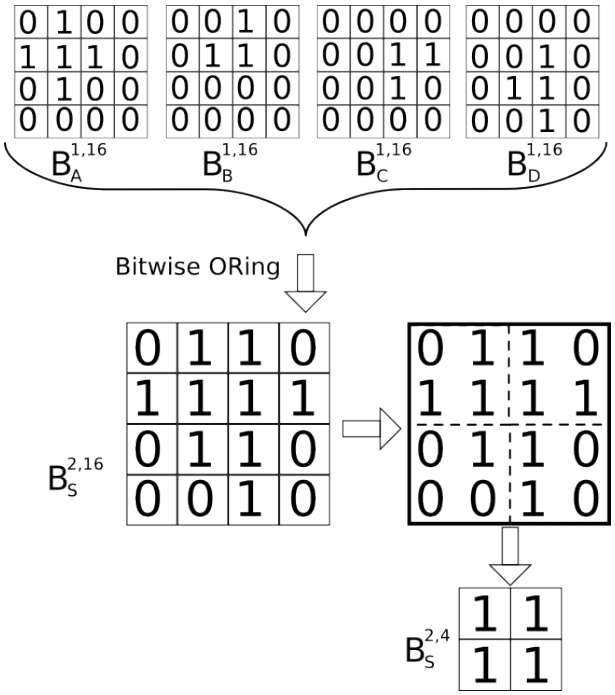

Figure 4: Constructing $B_S^{2,16}$ = 0110 1111 0110 0010 and $B_S^{2,4}$ = 1111

4.2 RASTER routing

For RASTER to be able to make a forwarding decision, each node needs to store certain bitmaps. RASTER(a) defines the algorithm, where each node s maintains bitmaps about its neighboring topology within a radius of a: Each node s keeps $B_s^{1,r_{max}}, B_s^{2,r_{max}}, \ldots, B_s^{a,r_{max}}$ and addition-

ally $B_k^{1,r_{max}}, \ldots, B_k^{a-1,r_{max}}$ for all neighbors k, where r_{max} is the maximum resolution.

The storage amount for each node is therefore $a \cdot r_{max}$ bits for its own bitmaps plus $k \cdot (a-1) \cdot r_{max}$ for its neighbors, where k is the out-degree, i.e. number of neighbors. Assuming that each node has a logarithmic number of neighbors ($k \in \mathcal{O}(\log n)$), the total storage cost for one node is $(a + \log n \cdot (a-1)) \cdot r_{max}$ bits.

The general approach of the routing decision procedure is as follows. When a node s receives a query for a key k, it first identifies the corresponding bin in the system's hash space, i.e. the bit in the bitmap, b_0. The node then forwards the query to its neighbor k^*, which (1) can reach the closest bin to b_0 and (2) following the minimum number of hops.

4.2.1 Straightforward implementation

Upon receiving a query node s searches $B_s^{1,r_{max}}, B_s^{2,r_{max}}, \ldots, B_s^{a,r_{max}}$ to find the closest bit to b_0 set to 1. If is b_0 found in more than one bitmap, choose the one with the smallest radius H. RASTER stops looking when the bit representing the destination bin is set. This means that there is a neighbor which can reach the closest bin in $H-1$ hops. To find this neighbor we look through $B_k^{H-1,r_{max}}$ for all neighbors k until one of them, k^*, has set the bit b_0 to 1. This is the neighbor to which the query will be forwarded to. Wang et al. [11] claim, that this approach had a significant drawback: Searching through all the bitmaps for the bit closest to b_0 is expensive. The *real* RASTER(a) implementation solves this problem by maintaining and searching through different resolution bitmaps. We will survey this claim in Section 5.

4.2.2 RASTER(a) implementation

The idea of this implementation compared to the straightforward one is to keep multiple resolutions for each bitmap. First you determine where the queried key resides for all maintained resolutions. Then, instead of searching all bitmaps with resolution r_{max}, different resolution bitmaps are searched for the bit closest to b_0. See Figure 5 for a visual representation.

4.2.3 Processing overhead

In an adaptive DHT network with n nodes and an out-degree of $k \in \mathcal{O}(\log n)$ which implements RASTER(a) maintaining l resolutions, has the following overheads [11]: To make the *forwarding decision* $\mathcal{O}(\log n)$ bitwise operations are necessary. The cost of *exchanging bitmaps* is $\Theta(a \cdot r_{max} \cdot \log n)$ bits and the cost of storing this information is $\Theta(a \cdot l \cdot r_{max} \cdot \log n)$ bits.

5. ROUTING DECISION PERFORMANCE COMPARISON

In this section we will compare the decision process' performance of the straightforward implementation and the more elaborate RASTER(a). These benchmarks written in Java were conducted on a regular consumer laptop.[2]

[2]Intel Core2Duo T7500 @ 2.21 GHz, 2 GiB RAM, Ubuntu 9.10, source code found at [3]

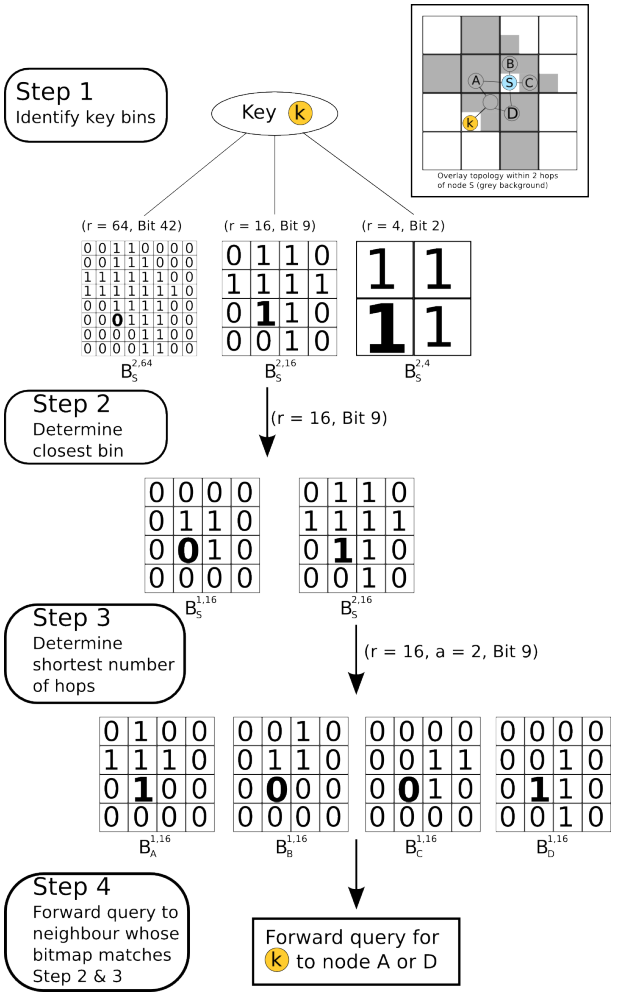

Figure 5: Node S' forwarding decision following RASTER(2): (1) Identify the destination bin's bit for every resolution (bold bits). (2) Find the largest resolution where this bit is set ($r = 16$). (3) Find smallest radius for which this bit is set ($a = 2$). (4) Find neighbor who can reach the destination bin in $a - 1 = 1$ hops.

5.1 Benchmarks

In a DHT with 80,000,000 nodes, an average out-degree of 50, a maximum resolution of 100 and a radius of 3, the straightforward implementation (SFWD) takes on average 27 milliseconds to perform 1 million routing decisions. The "faster" RASTER(3) takes 39 ms (with RASTER(3) using 3 additional scaled down resolutions by halving). In the following paragraphs we will look at how both algorithms react to parameter changes. The fixed parameters for the following benchmarks have the values mentioned at the beginning of this paragraph.

If we add more nodes to the DHT, resulting in more average overlay neighbors, both SFWD's and RASTER(a)'s time slowly increases. SFWD's time to calculate forwarding decisions rises faster and surpasses RASTER(a) at 83 neighbors (see Figure 6).

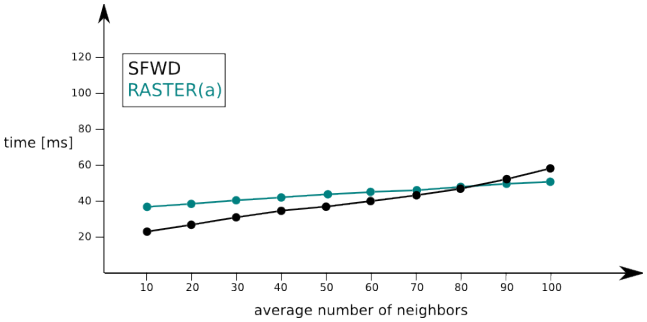

Figure 6: Performance impact of changing the number of neighbors.

By decreasing the bitmaps' resolution, SFWD's time to make a lookup decision decreases greatly, increasing the resolution increments this time. SFWD is thus sensitive to a resolution change. RASTER(a) is much less sensitive to this change, it begins to outperform SFWD at a resolution of just over 300 bits (see Figure 7).

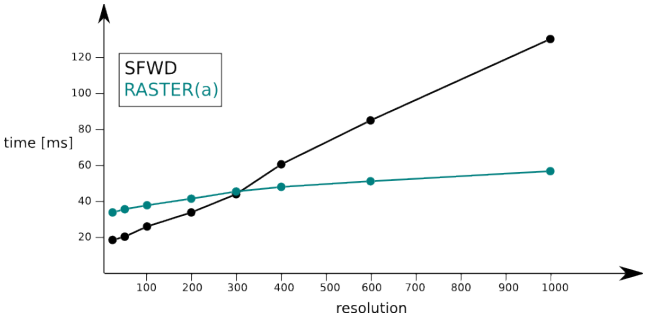

Figure 7: Performance impact of changing the resolution.

Altering the radius also results in SFWD's performance scaling slightly worse than RASTER(a)'s. See Figure 8 for a visual representation.

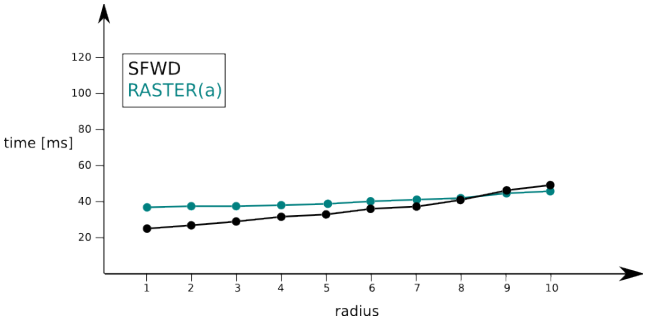

Figure 8: Performance impact of changing the radius.

5.2 Results

The result of these inquiries are that the RASTER(a) implementation is not generally faster than the straightforward one, although the former is of the order of $\mathcal{O}(\log r)$ and the latter is in $\mathcal{O}(r)$ and the performance of both is impacted by parameter choices.

The more interesting observation, however, is that RASTER(a) is a rather premature optimization. SFWD takes 27 ms to make 1 million forwarding decisions, which is approximately 40 million packets per second. Assuming a packet size of 1 kB this corresponds to 40 GB/s. The forwarding decision algorithm is thus — for moderately large resolutions r — not the bottleneck of systems implementing the RASTER protocol. For higher resolutions, the security issues raised in the following section are likely to be a bigger concern than performance.

6. RASTER SECURITY

One of RASTER's concerns, which is ignored in [11], is the aspect of security. In this section we will survey the impact of faulty or malicious nodes on RASTER.

6.1 Assumptions

Our inquiries are based upon these basic assumptions: (1) A malicious node is able to join the DHT network. In a decentralized DHT it is due to the absence of a central authority inherently difficult to prevent a malicious node from joining the network. (2) The malicious node is able to manipulate data (specifically change a bitmap), which resides on its system and by forwarding it to another node also modifying this node's bitmap data. (3) The malicious node tries not to be detected by other nodes. The malicious node therefore needs to adhere to the protocol's specification insofar as other nodes wouldn't get suspicious and ignore the malicious node.

6.2 Vulnerabilities

To understand RASTER's vulnerabilities we need to focus on its characteristics. One of them is that each node keeps special routing tables, the so called bitmaps. These bitmaps do not only contain information about direct neighbors but of neighbors of degree a. This is the main weakness of RASTER.

6.2.1 More neighbors

In an attack a malicious node could simply manipulate its bitmaps, so that it appears that it is able to reach a lot *more* neighbors within a radius a than it actually can. As a consequence more requests than usual will be forwarded to this node. The malicious node can then perform a series of attacks such as dropping the request, delaying it, sending a wrong response or compromise user's privacy by logging all activity. Possible attacks are the *Sybil* attack [2], where an attacker needs less nodes joining the DHT to get a certain control over it as nodes using bitmaps have a greater manipulation capacity. The same applies for the *Eclipse* attack [8]. Possible solutions to this attacks are listed in [10] and are outside the scope of this paper.

To understand how many additional requests a node receives, we need to discuss how the parameters a and r are best chosen. To have as much information entropy, i.e. encoded information, as possible in a bitmap half of its bits should be set to 1 and the other half to 0. Then we have to define a maximum influence factor x, that is the number of nodes one malicious node can influence. The radius is then chosen as

$$a = \lfloor \log_k 2 \cdot x \rfloor$$

where k is the average number of neighbors. The number of nodes a malicious node can influence is therefore bounded by $\frac{k^a}{2}$. As the resolution is chosen such that half of the bits are set to 1, an attacker can only set the other half to 1 as well, which corresponds to 50% of the nodes an attacker can possibly reach at the maximum radius.

The resolution is than chosen as

$$r = 2 \cdot k^a$$

to ensure that approximately half of the bits are set to 1.

If you have sufficient knowledge about the nodes participating in the DHT you can choose a threshold of maximum 1s set in order to detect and subsequently ignore malicious nodes. A malicious node could then always set as many bits to be just under the threshold and will thus not be detected. Thresholds work especially good if all nodes have very similar service capacities as the threshold is then just a fraction above the ideal 50% of 1s. A malicious node could still undercut this threshold but it can only influence fewer additional nodes than in a DHT with a diverse service spectrum.

6.2.2 Fewer neighbors

The contrary, i.e. manipulating its own bitmap so that it appears that the malicious node can reach *fewer* nodes within a radius a, is also a problem. By pretending that a node cannot reach any or very few neighbors, it avoids being selected to forward queries. A node therefore has to forward almost no routing decisions for other neighbors but its requests are granted without any objection. This phenomenon is called *free riding* or *leeching* and is discussed in detail in *Free Riding on Gnutella* [1]. This not only leads to inequality among nodes but it also compromises a DHT's security by diminishing the availability.

6.3 RASTER vs. greedy security

The difference between RASTER's bitmaps and greedy's normal routing tables is that in the former one node has a much larger manipulation influence radius than the latter. Specifically one node in a DHT with greedy routing can only influence its direct neighbors by manipulating its routing table. Thus the number of possibly manipulated nodes in a greedy routed DHT with n participants is $\mathcal{O}(\log n)$. Contrary to that a node in a DHT implementing RASTER's bitmaps, depending on the radius a, is theoretically able to manipulate a lot more nodes: $\mathcal{O}(\log^a n)$. The number of nodes a malicious node can manipulate, grows *exponentially* with the radius a.

To make this contrast more obvious let us look at the following example. Suppose we have a DHT with $n = 1000$ nodes and an out degree of $k = 5$. If using greedy routing, one node can manipulate just 5 nodes', i.e. its direct neighbors, routing decisions which corresponds to 0.5% of all the nodes. However, if the DHT uses RASTER(4), one malicious node could potentially influence the forwarding behavior of $5^4 = 625$ nodes which corresponds to 62.5% of all nodes in the DHT. One single node is able to regulate more than half of the DHT.

The security impact can be bound by restricting a as suggested in Section 6.2.1 or by comparing the number of set bits to network topology statistics.

7. CONCLUSION

We discussed the benefits of using adaptive DHT networks in contrast to deterministic DHTs. It is however suboptimal to use algorithms, which were designed for deterministic DHTs, for adaptive networks. As an example we explained the Chord algorithm and we described that the RASTER algorithm, which was specifically devised for adaptive DHTs, has many advantages over greedy based strategies. We then compared the performance of both RASTER implementations and concluded that the choice of forwarding algorithm does not really matter. Regarding security, algorithms, in which one node has a lot of information about its neighboring topology and exchanges this information with them like RASTER, are inherently more susceptible to malicious or faulty nodes. We have seen, that one malicious node in RASTER can manipulate up to $\mathcal{O}(\log^a n)$ additional neighbors' routing decisions. By selecting the parameters carefully it can be reduced to $\frac{k^a}{2}$.

8. REFERENCES

[1] E. Adar and B.A. Huberman: *Free riding on gnutella*, In First Monday, pages 2–13, Citeseer, 2000

[2] J. Douceur: *The sybil attack*, In Peer-to-Peer Systems, pages 251–260, Springer, 2002

[3] O. Gasser: *Source code used to compare performance of straightforward implementation and RASTER(a)*, `http://projects.net.in.tum.de/downloads/raster.tgz`

[4] J. Kleinberg: *The small-world phenomenon: an algorithm perspective*, In Proceedings of the thirty-second annual ACM symposium on Theory of computing, pages 163–170, ACM New York, NY, USA, 2000

[5] G.S. Manku and M. Naor and U. Wieder: *Know thy neighbor's neighbor: the power of lookahead in randomized P2P networks*, In Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, pages 54–63, ACM New York, NY, USA, 2004

[6] P. Maymounkov and D. Mazieres: *Kademlia: A peer-to-peer information system based on the xor metric*, In Peer-to-Peer Systems, pages 53–65, Springer, 2002

[7] A. Rowstron and P. Druschel: *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*, In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pages 329–350, Citeseer, 2001

[8] E. Sit and R. Morris: *Security considerations for peer-to-peer distributed hash tables*, In Peer-to-Peer Systems, pages 261–269, Springer

[9] I. Stoica and R. Morris and D. Karger and M.F. Kaashoek and H. Balakrishnan: *Chord: A scalable*

peer-to-peer lookup service for internet applications, In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pages 160, ACM, 2001

[10] G. Urdaneta and G. Pierre and M. van Steen: *A Survey of DHT Security Techniques*, Citeseer, 2008

[11] C.C. Wang and K. Harfoush: *RASTER: a light-weight routing protocol to discover shortest overlay routes in randomized DHT systems*, In Proceedings of the 12th International Conference on Parallel and Distributed Systems, pages 553–560, 2006