SECURITY PROBLEMS AND THEIR UPSHOTS IN ROUTING PROTOCOLS OF DHT BASED OVERLAY NETWORKS

[1] ANIL SAROLIYA, [2] VISHAL SHRIVASTAVA

[1] M. Tech. Scholar, Department of Comp. Science, ACEIT, Jaipur, Rajasthan, INDIA

[2] Asst. Professor, Department of Comp. Science, ACEIT, Jaipur, Rajasthan, INDIA

E-mail: anilsaroliya@gmail.com, vishal500371@yahoo.co.in

ABSTRACT

Distributed hash tables (DHTs) are a very interesting research topic in the area of P2P overlay networks; such networks are becoming very popular in applications like file sharing. The purpose of the Distributed Hash Table is providing the way to search the resources (especially files) within a P2P network. A DHT protocol typically provides a single function to the P2P application: provide a key and find the node (or may be nodes) which is liable for such key [1][3]. All other functions (such as actually retrieving the resource or storing the resource on the node responsible for it) are provided by higher layers of the P2P application. In such paper our target is to find the security issues and resolve them on existing routing protocols of such networks. The Chord [4] (a DHT protocol) is chosen as the target protocol for various reasons it will be covered in this paper.

Keywords: *Peer-To-Peer Overlay Networks, Distributed Hash Tables, Routing, Security*

1. INTRODUCTION

1.1 Distributed Hash Table's

DHT is capable to accomplish two of our main needs. DHT is the distributed data structures that hold the key and the value as a pairs in the fully distributed way. It also puts every key-value couple only on the single or limited node. To decide on which node an exact pair has to be stored we require a mapping mechanism. The joining and disjoining a node does not cause the remapping of all the keys. A specific hashing mechanism (consistent hashing) is used in DHT to map the key. Such hashing separates the key in many parts. This process employs the distance concept to map a key to a certain node. Distance is a logical aspect and which is not related to or bounded to physical distance of the nodes in P2P network. In this network the node which is actually in England might be closer to the node in Japan than a node available in the same region. The mapping function will be in use when the insertion of the fresh key-value pair into the hash table will take place and also as we desire to search the key. This function utilizes only the key to decide the appropriate node which will hold the pair. After that, if the same key will be asked, then the above mapping function will resolve the place where the key is available, this process makes the recovery of the value quicker.

Distributed hash table (DHT) protocols allow resources to be located quickly in decentralized distributed systems. Resources can include things such as files, directory entries, discussion messages, or any other type of object that can be stored on and retrieved by nodes in a distributed system. A DHT consists of a group of participating nodes, where each node maintains a small amount of information about a subset of other nodes in the system and routes lookup requests through the system towards their destinations. Each resource has a key associated with it. Given a key, a DHT can locate the node responsible for the associated resource quickly, typically within O(log n) hops, here n is the number of available nodes in the system. The number of other nodes in the system that each node needs to be aware of is also typically O(log n). Well known DHTs that received a huge amount of concentration consist of CAN, Pastry, and Chord.

1.2 Chord Routing Algorithm

Individual numerical *identifiers* are provided for both nodes and keys in Chord protocol. To get the key's identifier just hash that key by particular hash function which is utilized by every nodes of

the system that returns *m* bit integers. The node finds its identifier by hashing of its IP address. Now these identifiers are ordered on an identifier circle (ring) modulo 2^m. Every key's value is assigned to the first node whose identifier is equal to or follow that key's identifier in the ring. This aspect is illustrated following figure 1.

In the Chord ring shown in figure 1[4], the hash bit length *m* is 6. There are 10 nodes in the network (shown with N prefixes followed by the node's identifier) and 5 keys (shown with K prefixes followed by the key's identifier) are being stored. Each key is shown being stored on the first node that succeeds the identifier of key in circle, as indicated by the arrows.

Every node stores some routing information to locate nodes which are legally dependable for keys. In Chord, this routing table is called a "finger table."

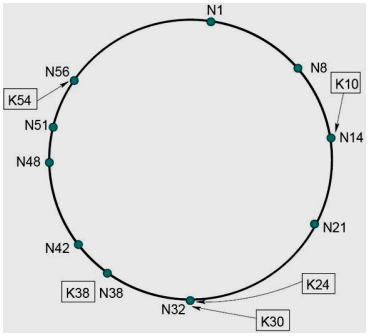

Figure 1: An illustration of keys mapping to nodes.

The Chord finger table for a node with identifier *id* contains *m* entries (0 to *m*-1 entries). For finger table entry *i*, the node stored in that entry is the first node whose identifier succeeds $id + 2^i$ (mod 2^k). It is possible (and often probable) to have duplicate entries in the finger table. Figure 2 shows a sample finger table with an illustration of how the finger table is derived for node N8. N8's last finger table entry should be the node that succeeds $8+2^5$. This node is N42, so a reference to N42 is stored in the last finger table entry of N8's finger table. The rest of the finger table entries are filled in with the same process for i = 0, 1, 2, 3, and 4.

As figure 2 illustrates, each node only has information about a subset of the nodes in the overall system. As the system gets much larger, the number of unique nodes in each node's finger table becomes a smaller fraction of the overall number of nodes. The size of the finger table has been shown by [4] to be O(log n) where *n* is the number of nodes in the system. The advantage of the finger table is that when performing a lookup we can jump about half of the remaining distance between the node doing the routing and the node responsible for the key.

This divide and conquer approach to routing lookup requests has been shown by [4] to use O(log n) hops for each route. The algorithm for routing a lookup request from a node is simple: forward the request to the last finger table entry that precedes the identifier of the key.

The node preceding the destination node will detect that the key falls between itself and its successor and return information about its successor to the node performing the lookup. Figure 3 presents an example of the route a lookup request might take through a Chord network. In this figure, N8 is performing a lookup request for key K54. For a new node to join a Chord network, it needs to know of any one node that is already in the network. Finding a node that is already in the network is done out of band. The joining node will then use this "bootstrap" node to perform a lookup on its own identifier. The node returned by this lookup will be the new node's successor in the Chord ring. The new node will send a message to its successor notifying it that it is now that node's predecessor and the successor will inform its previous predecessor that the new node is now its successor. The new node will then use its successor to perform the appropriate lookups to fill in its finger table.

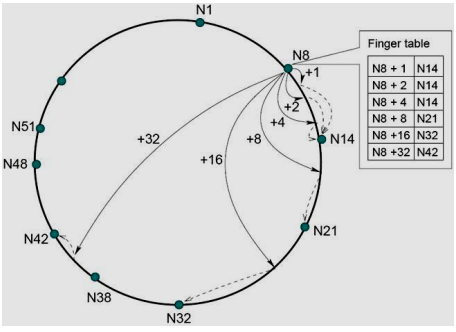

Figure 2: An example finger table, taken from [4].

Since nodes will be joining and leaving continuously, each node needs to periodically re-perform these lookups in order to keep its finger table up to date.

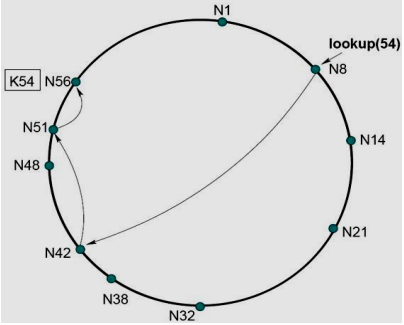

Figure 3: An example of the route taken by a lookup in a Chord network, taken from [4].

1.3 Chord Attack Vulnerabilities

Since DHT lookup requests rely on other nodes in the system to follow the protocol correctly, they are vulnerable to several types of attacks ([4]; [5]).

One category of attack is routing attacks, and this is the category that this paper will focus on defending against. A routing attack occurs when a node intentionally drops lookup requests or forwards the lookup request to another node in a manner that violates the protocol specification. Examples of incorrect forwarding would be to forward the request to nodes further away from the destination, to random nodes, or to other colluding malicious nodes. Colluding malicious nodes might run a separate Chord partition or a "sub-ring" in a real Chord network and capture lookup requests and forward them into this sub-ring. This attack makes it seem as if lookup requests are being forwarded correctly and it could even cause nodes joining the system to unknowingly join the malicious partition.

Another category of attack is an attack where the node responsible for a key returns incorrect values for that key. It is difficult for an attacker to target specific keys in Chord since a malicious node's identifier is a hash of its IP address which forces a node into a specific area of the network and makes it easy for other nodes to verify that a node is using its correct identifier. It is left to higher levels in the P2P application to verify that the retrieved data from nodes is correct once the lookup process completes successfully. Chord allows for a key's corresponding value to be stored on multiple nodes (called *replicas*) by using multiple hash functions to obtain multiple identifiers for keys. This paper will not focus on attacks where nodes responsible for keys misbehave; instead we focus on preventing malicious nodes from keeping lookups from reaching the node(s) responsible for them.

Yet another method of attacking a Chord network is for a bootstrapping node to bootstrap a joining node into a malicious network instead of the intended network. Bootstrapping is out of band, and there is little that can be done if a malicious node is used to bootstrap. We will therefore assume that the node used for bootstrapping is trusted.

2. PROPOSED DEFENSE MECHANISMS

To mitigate routing attacks on Chord, we propose the following major changes to the protocol:

- Instead of lookup requests being forwarded from node to node, the node performing the lookup will directly contact each node and request the next hop on the route to the destination.

- Each hop will be verified for probable correctness by checking the numerical difference between node identifiers in the hop to statistical information about network density derived from the finger table of the node performing the lookup.

- If a hop is determined to be invalid, the node performing the lookup will backtrack to the previous node on the route and ask for a different finger table entry.

Each of these changes is described in more detail in the following sections.

2.1 Source Node Routing

In the Chord protocol, a node performing a lookup forwards the lookup request to the closest preceding node in its finger table. Instead of forwarding our lookup request out into the un-trusted network, we will ask each hop in the route for the next hop ourselves. This is possible in overlay networks since we can establish a "direct" overlay connection to any node on the physical network. This change is straightforward to implement.

When we perform lookups from the source, when we detect a malicious node along our route we can go back to the last good node on that route and ask for an alternative next hop. Again, we

cannot rely on other nodes to perform these actions since other nodes are not trusted.

2.2 Route Hop Verification

Route hop verification is the most important change being made to the protocol. The goal of hop verification is to answer this question: node a returned a reference to node b as the next hop on the route to some key. Is this hop correct?

The main idea is to look at the distance between the identifier of the next hop returned by node A and the "pointer" used by node A for the finger table entry of that next hop and determines if this distance is likely given the density of the network. A finger table entry pointer for finger table entry i of a node with identifier id is id + 2i (mod k). This is the identifier that a node looks up when it is filling in finger table entry i. We know that the finger table pointer must fall between two nodes in the Chord ring, so the distance between an entry's pointer and the identifier of the actual node stored in that entry is less than the distance between that node and its predecessor in the ring. By comparing the numerical distance between the entry pointer and the entry node's identifier (the dashed line in figure 2) to the average numerical distance between nodes, we can determine how likely it is that a node is using a proper node for a particular finger table entry.

Each node will estimate the average numerical distance between nodes in the ring from its own finger table. Nodes in this modified version of Chord will store additional information about its finger table entries for this purpose. When performing periodic finger table updates, nodes will query the nodes in its finger table for the identifiers of those nodes' predecessors and successors. This gives a node up to two unique distance samples per finger table entry. From these samples each node will compute its estimate of the average distance between nodes and the standard deviation of those distances from the average.

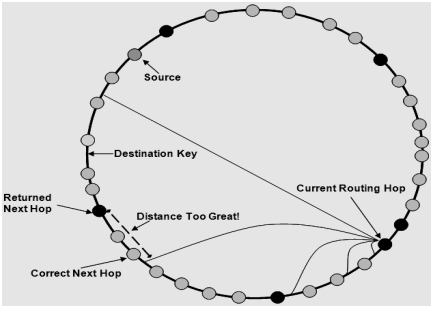

Figure 4: An illustration of how hops are verified.

The green node is the source node, the yellow node is the destination node, blue nodes are uncompromised nodes in the network, and red nodes are compromised nodes in the network.

If the distance between a finger table entry's pointer and the entry node's identifier is greater than the average distance between nodes plus a parameter times the standard deviation of the average distance between nodes, we will consider the hop invalid. Otherwise we consider it valid. Figure 4 above illustrates the hop verification process. In this diagram, the green node represents the source node and the yellow node represents the destination node (the successor to the destination key.) Blue nodes are nodes that are uncompromised and are correctly participating in the protocol. Red nodes are malicious nodes that have formed a sub network in order to capture lookup requests and forward them among malicious nodes.

2.3 Defense Strategy

The idea of source routing, detecting malicious nodes on the path to the destination, and routing around those malicious nodes should get us to the valid destination for a given key. If a node is simply dropping lookup request, we can backtrack and go around it and continue to the destination. If node A refers a lookup requests to incorrect node B, we can verify that that node B should not be present in the finger table entry returned by node A. If a group of malicious nodes form a sub ring in the Chord network and run the Chord protocol among themselves and only return routing references to other malicious nodes, the malicious network will have a lower node density than the rest of the Chord network and we will detect this as higher than expected node distances when we query the malicious nodes. We can then route around these malicious nodes.

3. GENERAL IDEA OF SYSTEM

The modified Chord protocol is implemented in a simulator written in Java. The simulator allows the user to run simulated Chord networks of varying sizes. The user is able to decide how many nodes will be compromised. The user will have the ability to specify the standard deviation parameter, which is described in section 2.2. While the system should have the ability to test for malicious nodes that drop packets and return

random incorrect next hops, the more interesting test the system will run will be when a group of nodes are colluding by running a sub-ring of malicious nodes and only returning other nodes within this sub-ring as hops when receiving a lookup request. The software simulator will perform lookups of random identifiers from random, non-compromised nodes in the system. The user will have the ability to specify how many tests will be performed.

4. ASSUMPTION AND TESTS

This section contains a list of assumption about the proposed system that we plan to investigate, and how those assumptions will be tested.

i. Nodes performing lookups should be able to route around malicious nodes that simply drop lookup requests. This will be tested by varying the fraction of malicious nodes that drop lookup requests, performing a number of lookups of random keys from random nodes, and recording the lookup success rate. The lookup success rate will then be plotted against the fraction of malicious, lookup request dropping nodes.

ii. Nodes performing lookups should be able to detect non-colluding malicious nodes that return false next hops. This will be tested in the same fashion as assumption1, except the malicious nodes will be set up to return random, incorrect nodes during the lookup requests instead of simply dropping the packet. The same data will be plotted. This has the effect of testing the verification and the backtracking algorithm, but this type of attack should be easier for the verification algorithm to detect than the attack in the next assumption.

iii. Nodes performing lookups should be able to detect colluding malicious nodes that are running a sub-ring and are returning references to other malicious nodes during the lookup process. Again, this will be tested in the same fashion as assumption1 and 2, except the malicious nodes will be running an alternate finger table that consists of only malicious nodes and using that finger table during lookup requests. This has the effect of testing the verification algorithm under the attack method that is the most difficult to detect.

As the number of nodes in the system increases, the success rate in avoiding all three attacks described above should increase. The data taken from the tests for i, ii, and iii will test this assumption. tests will be performed.

5. CONCLUSION & FUTURE SCOPE

Mischievous nodes in DHT's can introduce severe interruptions, even when they only exist in small numbers. Several security concerns must be targeted in order to use DHT's in situations where users cannot be trusted. In this paper, we proposed a mechanism for mitigating the effects of one of those concerns: routing threats.

The future work of regarding this aspect is, the technique discussed here should transfer to other DHTs protocols that make use of constrained routing, and can serve as a crucial piece to a total security solution.

REFRENCES:

[1] Heinbockel, W., and Kwon, M.: Phyllo: A peer-to-peer overlay security framework. The First Workshop on Secure Network Protocols (NPSec), Boston, MA (2005)

[2] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: Proc. ACM SIGCOMM'01, San Diego, CA (2001)

[3] Saroliya Anil, Shrivastava Vishal: Analysis of Routing Attacks in Peer to Peer Overlay Networks, Jaipur, In: National Conference on "Recent Trends in IT: Opportunities and Challenges", S.S. Jain Subodh MCA Inst., Kukas, Jaipur (2010)

[4] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proc. ACM SIGCOMM'01, San Diego, California (2001)

[5] Wallach, D.: A survey of peer-to-peer security issues, International Symposium on Software Security, Tokyo, Japan (2002)