Oriental Journal of Computer Science & Technology Vol. **3**(2), 349-353 (2010)

Analysis of routing attacks in peer to peer overlay networks

ANIL SAROLIYA[1] and VISHAL SHRIVASTAVA[2]

[1]Department of Computer Science, Arya College of Engineering and IT, Jaipur (India).
[2]Department of CS, Arya College of Engg. & IT, Kukas, Jaipur (India).

(Received: May 23, 2010; Accepted: July 03, 2010)

ABSTRACT

Peer-to-peer (P2P) systems are distributed systems in which nodes act as peers, such systems are becoming very popular in applications like file sharing. In this kind of architecture, security in each transaction is fundamental requirements. The aim of a Distributed hash tables provides the method for locating resources (generally files) within a P2P network. In this paper our target is to analyze the routing attacks on existing protocols of such networks. Chord is preferred as the target DHT protocol for various causes which will be discussed in the paper. Routing attacks analysis finds the vulnerabilities of existing protocols and anticipates a defense mechanism which is discussed herewith.

Key words: Routing attacks, peer to peer overlay networks.

INTRODUCTION

Overview of DHT and CHORD

Distributed Hash Table

DHT is capable to accomplish two of our main needs. DHT is the distributed data structures which hold key and value pairs in a fully distributed manner. It also puts each key-value pair on a single or limited node only. To decide on which node a specific pair has to be stored we need a mapping method. In addition, joining and disjoining a node does not cause all the keys to be remapped. A specific hashing mechanism (consistent hashing) is used in DHT to map the key. Such hashing separates the key space into partitions. This process employs the distance concept to map a key to a certain node. Distance is a logical aspect and not required to be related to physical distance of nodes in network. In P2P network a node which is actually in France could be closer to a node in China than a node in the same country. The mapping function is being used when we need to insert a new key-value pair into the hash table and also when we want to find the key. This function uses the key itself to determine the node which will store a pair. Then, when the same key is being queried, the same mapping function can determine the place which the key is being stored and therefore make retrieval of the value faster.

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, where n is the number of 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). Popular DHTs that have received a great deal of attention include CAN, Chord, and Pastry.

Chord Routing Algorithm

In Chord, both nodes and keys are given

numerical *identifiers*. The identifier for a key is obtained by hashing that key with some hash function that is used by all of the nodes in the system which returns integers of some bit length *m*. A node is assigned an identifier by hashing its IP address. Nodes and keys are then arranged in an *identifier ring* modulo 2^m. Each key's value is stored on the first node with an identifier equal to or following that key's identifier in the ring. This aspect is illustrated following Fig. 1.

In the Chord ring shown in Fig. 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 that key's identifier in the ring, as indicated by the arrows.

In order to find nodes that are responsible for keys, each node has to store some routing information. In Chord, this routing table is called a "finger table."

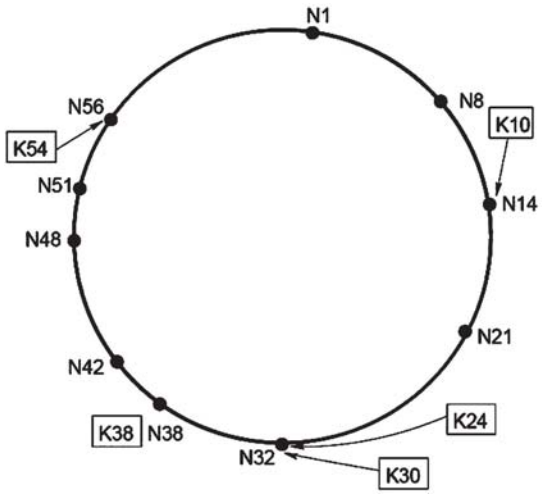

Fig. 1: An illustration of keys mapping to nodes

The Chord finger table for a node with identifier *id* contains *m* entries, numbered from 0 to *m*-1. For finger table entry *i*, the node stored in that entry is the first node whose identifier succeeds $id + 2^i \pmod{2^k}$. It is possible (and often probable) to have duplicate entries in the finger table. Fig. 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 Fig. 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.

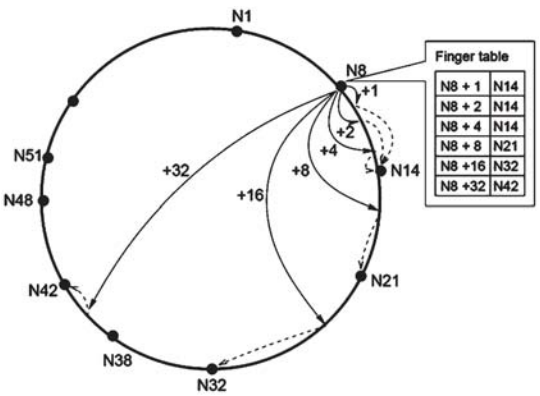

Fig. 2: An example finger table, taken from [4]

Fig. 3 shows 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.

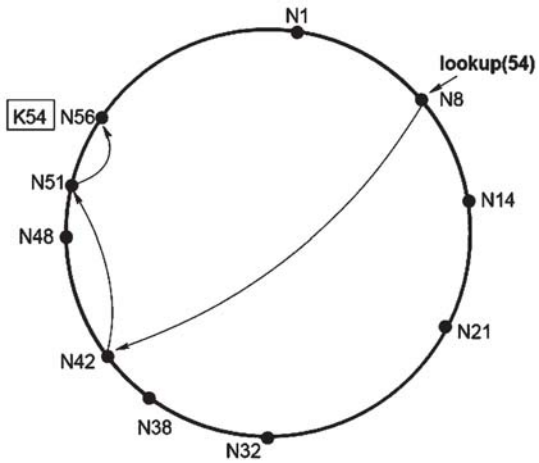

Fig. 3: An example of the route taken by a lookup in a Chord network, taken from[4]

The new node will then use its successor to perform the appropriate lookups to fill in its finger table. 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.

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.

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.

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.

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 $id + 2^i \pmod{k}$. 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.

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

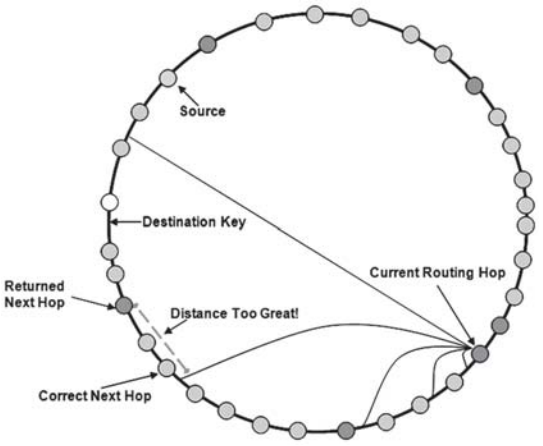

Fig. 4: An illustration of how hops are verified.

valid. Fig. 4 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.

CONCLUSION

Through malicious node can be breach DHT's security, 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. Security in structured p2p networks is difficult because of their fully distributed nature, but we have shown that routing security can be greatly improved using only a relatively small amount of locally known information.

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.

REFERENCES

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: Discovering and Recovering from Routing Threats in Distributed Hash Tables of P2P Networks, Jaipur, In: All India Conference on "Recent Innovation in Software and Computer", ACEIT, 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).