Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.4, No.4, 2013

Distributed Hash Tables in P2P Network: Detection and Prevention of Threats and Vulnerability

Mohammad Naderuzzman (Corresponding Author)
Department of Computer Science & Engineering
Dhaka University of Engineering & Technology,
Gazipur, Dhaka
E-mail: nader_u@yahoo.com

Dr. Md. Nasim Akhtar
Department of Computer Science & Engineering
Dhaka University of Engineering & Technology,
Gazipur, Dhaka
E-mail: nasim_duet@yahoo.com

Abstract
Currently the peer-to-peer search focuses on efficient hash lookup systems which can be use in building more complex distributed systems. These system works well when their algorithms are executed in right direction but generally they don't consider how to handle misbehaving nodes. In our paper we considers different sorts of security problems which are inherent in peer-to peer systems based on distributed hash lookup systems. We examine different types of problems that this kind of systems might face, taking examples from existing systems. Here we propose some design principles for detecting as well preventing those problems.
Keywords- Distributed hash lookup systems, verifiable system invariants, verifiable key assignment, Server selection in routing.

1. Introduction

Recently a great number of systems were built on top of distributed peer-to-peer hash lookup systems [6,9,10]. Keys lookups are performed by queries routing through a series of nodes; each of these nodes maintains a local touting table to forward query towards the node which is ultimately responsible for the key. These nodes can be used to store data, i.e. as a distributed hash table or may be file system [1,7]. Some researcher took advantage of other aspects of the lookup system, like the properties of lookup routing [8]. This is unfortunate that the architecture of many of these systems assume that the nodes involved in a system are trusted. In an intranet, such as inside a corporate firewall, the assumption of trust might be justified, but on an open network, like the Internet, still it may be possible to exclude un-trusted nodes with the help of a central certificate granting authority; whose solution was proposed by Pastry [6]. But there may be many situations in which it is not desirable to constrain membership of a peer-to-peer system. In situations like this, the system should be able to operate even though some participants are likely a malicious.

A kind of attacks on distributed hash tables causes the system to return incorrect data to the application. Fortunately, the authenticity and correctness of such data can be addressed by using techniques like cryptographic, for example self-certifying path names [3]. The techniques detect and ignore un-authentic data in the systems. This paper focuses on the those attacks that threaten the aliveness of the system by preventing participants from finding data. The main part of the paper is a series of examples of particular weaknesses in existing distributed hash algorithms. Our paper discusses potential defenses for few of these problems, and derives a set of general design principles from them and summarized in *Table 1.* All these principles are driven by the fact that any information obtained over the network can not be trusted and hence must be verified.

Sl.	Design Principles
1	Allow the querier node to observe lookup progress
2	Define verifiable system invariants by a node
3	Assign keys to nodes in a verifiable way
4	Server selection in routing may be abused
5	Cross-check routing tables using random queries
6	Avoid responsibility to a single point (node).

Table 1: Design Principles

2. Background

In general distributed hash tables consist of a storage API layered on top of a lookup protocol. Each lookup protocols consist of a few basic components:

1. a key identifier space
2. a node identifier space
3. rules for associating keys to a particular node
4. routing tables for each node that refer to other nodes
5. set of rules for updating routing tables as nodes joins and leave

Any lookup protocol maps a desired key identifier to the IP address of a node responsible for that key. A storage protocol is layered on top of the lookup protocol, then storing, caching, replicating, and authenticating of data are taking care of. CAN[5], Chord[9] and Pastry [6] all these protocol fits into this general framework.

In the lookup, routing is handled by defining by a distance function on the identifier space, such that distance can be measured between the current node and the desired key; the node responsible is defined to be the node closest to the key. Typically a lookup protocol has an invariant that must be maintained in order to guarantee that data can be found. As an example, in the Chord system, nodes are arranged in a one-dimensional identifier space; here the required invariant is that every node knows the other node that

immediately follows it in the identifier space. In case an attacker breaks this invariant, Chord system will not be able to look up keys correctly.

Similarly, in order to be sure that each piece of data is available, the storage layer will also maintain some invariants. In DHash [1], a storage API layered on Chord used by CFS, there have been two important invariants, first, it must ensure that the node which Chord believes is responsible for a key actually stores the data associated with that key. It is important that DHash maintain replicas of each piece of data because nodes can fail, and that those replicas be at predictable nodes. An attacker may potentially target either of these invariants.

3. Define Adversary Model

In this paper, the adversaries that we considered are participants in a distributed hash lookup system that do not follow the protocol correctly. Instead, by providing them with false information, they seek to mislead legitimate nodes.

We assumed that a malicious mode is able to generate a packet with arbitrary contents, including forged source IP, but that node is only able to examine packet's addressed to itself. i.e. malicious nodes are not able to modify communication or overhear between other nodes. A malicious node can only receive packets addressed to its own IP address means that an IP address can be used as a weak form of node identity. If any node receives a packet from an IP address, it can verify that the packet's sender owns the address by sending request for confirming that IP address. We consider malicious nodes conspire together, but each one is limited as above. This allows to gather additional data by an adversary and act more deviously by providing false but confirmable information.

Rest of the paper will examine different ways in which a malicious node can use these abilities to subvert the system.

4. Different Attacks and Defenses

This part of the paper organized into attacks against the routing, attacks against the data storage system and finally some general considerations.

We know that the first line of defense for any attack is detection of the attack. Many attacks can be detected by the node being attacked, because the nodes which are involved violating invariants or procedure contracts. However, once an attack has been detected, it is less clear what to do. A node may really be malicious or may be it have failed to detect that it was being tricked. So, our discussion focuses on the methods to detect and possibly correct in consistent information. Here we will see that achieving verifiability underlies all of our detection techniques.

4.1. Attacks on Routing

In a lookup protocol only the routing portion involves maintaining routing tables; it then dispatches requests to the nodes in the routing table. It is quite difficult to identify that the routing is correct in a distributed hash table. n existing system. There are considerable chances for an adversary to play in existing systems. This kind of attacks can be detected if the system *defines considers verifiable system invariants* and verify them. When an invariant fails, the system must have some recovery mechanism.

Incorrect Lookup Routing A single malicious node may forward lookups to an incorrect or non-existent node. Because the malicious node will be participating in the system's routing update in a usual way, it will appear to be alive and will not ordinarily be removed from the routing tables of other existing nodes. In this way re-transmissions of the misdirected lookups will also be sent to the malicious nodes.

Luckily blatantly incorrect forwarding can easily be detected. The querier knows that the lookup is supposed to get 'closer' to the key identifier at each hop. The querier should check, so that this attack can be detected. If this kind of attack is detected, the querier might recover by backtracking to the last good hop and may ask for an alternative steps which offers less progress.

For a querier node to be able to perform this kind of check, each steps of progresses must be visible to the querier. As an example, CAN proposes an optimization where each node keep tracks of the network RTTs to neighbor nodes and forward to the neighbors with the best ratio of progress to RTT. This proofs that queries are normally forwarded without consulting with the querier node. Thus in CAN, a querier node simply can't verify forward progress. So the querier node should be allowed to observe the lookup progress.

The malicious nodes may also may declare (incorrectly) that a random node is the node which is responsible for a key. Because the querying node may be far away in the identifier space, It may not know that this node is not the closest node in fact, which could cause a key to be stored in an incorrect node or may prevent the key from being found. This type of problem can be fixed in following two ways:
Firstly, the querier node must ensure that the destination node itself agrees that it is in fact a correct termination point for the particular query. In Cord system, the predecessor returns the address of the query's endpoint (i.e. 'successor') instead of the endpoint itself, which allows the attack possible. A malicious node may cause the query to undershoot the right successor, which may cause DHash to violate its storage location invariant. If the node that referred to is a good node, then it should not be responsible for this key and can generate an error.

Secondly, assignment of keys to a node should be in a verifiable way by the system. Particularly in some systems, keys are assigned to the node which is closest to them in the identifier space. Thus, to assign keys to nodes verifiably. it is sufficient to derive node identifiers in a verifiable way. In contrast this to CAN, that allows any dode to specify ots own identity. Which makes it not possible by another node to verify that

a node is validly claiming responsibility for a key. In some system, like Chord, gave an effort to defend against this by basing a node's identifier on a cryptographic hash of IP address and port. Since this needs to contact the node, it is easy to say if one is speaking to the correct node.

A long-term identities based on public keys may be derived by a system, which has performance penalties because of the cost of signatures, but would allow systems to have faith on the origin of messages and will validate of their contents. This means, public keys will facilitate the verifiability of the system. Particularly, a certificate with a node's public key and address can be used by new nodes to safely join the system.

Incorrect Routing Updates In a lookup system each node builds its own routing table by consulting other nodes, a malicious node may corrupt the routing tables of other nodes by sending them incorrect updates. Effect of these updates would cause good nodes to misdirect queries to inappropriate nodes or to non-existent nodes. If the system knows correct routing updates follows certain requirements, this can be verified. For example, in Pastry systems, updates require that every table entry has a correct prefix. Blatantly incorrect updates can easily be identified and dropped. Only after verifying itself that the remote node is reachable, the updates should be incorporated in a node's routing table.

By taking advantage of systems that allow nodes to choose between multiple correct routing entries, a more delicate attack would be eminent. For example, to minimize latency, CAN's RTT optimization allows precisely in order to minimize latency. A malicious node may take advantages of this flexibility and my provide nodes that are undesirable. For example, it may choose an unreliable node, node with high latency or even a fellow malicious node. This may not affect strict correctness of the protocol but it may affect applications that may wish to use underlying lookup system to find nodes satisfying certain criteria. For example, in Tarzan anonymizing network [2] it proposes the use of Chord as a way of discovering random nodes to be used in dynamic anonymizing tunnels. Any flexibility in Chord may allow adversary to bias the nodes chosen and may have to compromise the design goals of Tarzan. The applications of this should be aware that server selection in routing may be abused.

Partition For a bootstrap to happen, a new node wish to participate in any lookup system must contact some existing node. At the time of bootstrap, it is vulnerable of being partitioned into an incorrect network. For example, suppose a set of malicious nodes formed a parallel network, which are running the same protocols as the real and legitimate network. This type of parallel network is entirely internally consistent and may contain some data from the real network. Accidentally any new node may join this network and thus will fail to achieve correct results. Any malicious node might also cross-register in the legitimate network and may cause new participants to be connected to the parallel network even if they have a valid bootstrap node.

Malicious nodes may deny service by using partitions or may learn about the behavior of clients that it would otherwise be unable to observe. For example, let say a service was made available to publish

documents anonymously, at that time an adversary could establish a malicious system that shadows the real one but allows it to track clients who are storing and reading files.

Preventing a new node from being diverted into an incorrect network, the node must bootstrap via some sort of trusted source. Such trusted sources are likely be out-of-band to the system itself. At the time of rejoining to the system, a node can either use these trusted nodes or it may use one of the other nodes it has previously discovered in the network. However developing trust metrics for particular nodes can be risky in a network with highly transient nodes that lack any strong sense of identity. Via DHCP if a particular address is assigned, for example, sometime a node could be malicious but benign the next. Also in this case use, use of public keys may reduce the risk.

In case a node believes that it has successfully joined a network in the past, then the node can detect new malicious partitions by cross-checking with the history stored with it. A node can maintain a set of other node's information that it has used successfully in the past. So that it can cross-check routing tables by using random queries. Also by asking those nodes to do the same random queries and lastly comparing those results with its own. This way a node can verify whether its view of the network is consistent with the other nodes. Randomness is important because a malicious partition can not distinguish verification probes from a legitimate query that it would like to divert. On the contrary a node which has been trapped in a malicious partition may accidentally discover the correct network in this way, where the right network may be defined as the one which serve desired data.

4.2 Attacks on Storage and Retrieval

Any malicious node is able join and participate in the lookup protocol correctly, but will be denied the existence of data it was responsible for. It might also claim to actually store data when asked, but then refuse to serve it to clients. To handle this type of attack, the storage layer must imply replication. The replication should be handled in such a way that no single node is responsible for replication or facilitating access to the replicas; that node will be a single point of failure. So, the client must be able to determine independently the correct node to contact for replicas. This will allow them to verify that truly data is unavailable with all replica sites. Similarly, all nodes those are holding replicas must ensure that the replication invariant (i.e. at least n copies exist at all times) is maintained. If not so, a single node would be able to prevent all replication process to happen. This is to avoid single points of responsibility.

Nodes doing lookups should be prepared for the threat of possible malicious nodes as well. For this, it must consult at least two replica sites to be sure that either all of the replicas are bad or that the data is truly missing.

For example, a DHash does not follow this principle; here only the node immediately associated with the key will be responsible for the replication. Even, if the storing node performed replication, DHash will still

be vulnerable to the actual successor lying about the r later successors. As proposed in CAN, replication with multiple hash functions is one way to avoid this reliance on a single machine.

The attack can further be refined in a system which does not assign nodes verifiable identifiers. In this type of system, node can choose to become responsible for the data that it wishes to hide. Here DHash still are at risk, despite Chord having verifiable node identifiers, which is because the identifier was derived from a hash of node's IP address, port number and virtual node number. Because of a person in control of a node can run a large number of virtual nodes, still they effects some degree of choice in which data they wish to hide. IPv6 or sparsely used IPv4 networks may also allow to have access to many addresses by a single host.

4.3. Other Miscellaneous Attacks

Inconsistent Behavior If a malicious node presents a good face to part of the network, it would be more difficult to detect when it attacks. A malicious node may choose to maximize its impact by ensuring its behavior correctly for certain nodes. In the identifier space, one possible class would be nodes near it. Despite the fact that nodes that are distant see poor or invalid behavior, these nodes will not see any reason to remove the node from their routing tables. If queries must routed through close nodes before reaching the target node, this may not be a serious problem. However, most of the routing systems have their ways of jumping to distant points in the identifier space for speeding up queries.

Ideally, a distant node would be able to convince local nodes that 'locally good' malicious node is in fact a malicious. However, without public keys and digital signatures, it is not possible for a node to distinguish a report of a 'locally good' node being malicious. From a malicious report trying to tarnish a node which is actually a benign. On the other hand, this can be proven with public keys by requiring nodes to sign all of their responses, then a report would contain the incorrect response and the inappropriateness could be verified. Lacking this, every node must determine of its own as to whether another node is malicious.

Overload of Targeted Nodes It can attempt to overload targeted nodes with garbage packets because and adversary can generate packets. It is a standard denial of service attack and not a subversion of the system. This would cause the node to appear to fail and hence the system will be adapted to this as if the node failed in some normal manner. A system must use some degree of data replication so that it can handle even the normal node failure case. The attack will be effective if the replication is weak or if the malicious node is one of the replicas or may be colluding with some of the replicas.

Denial of service attacks impact can be partially mitigated by ensuring that the node's identifier assignment algorithm assigns identifiers to nodes randomly with respect to network topology. Additionally, the replicas should be located in such locations where they will be physically disparate. These would prevent a localized attack by preventing access to an entire portion of key space. If an adversary wishes to shut out an entire portion of the key space, it should have to flood packets all over the Internet.

Rapid Joins and Leaves Nodes join and leave the system, the rules imply that ne nodes must obtain data from replicas which was store by nodes that left the system. In order for the lookup procedures to work correctly, this rebalancing is required. A malicious node may trick the system into rebalancing which unnecessarily causes excess data transfer and control traffic. Which in turn reduce the efficiency and performance of the system. This kind of attack will work best if the attacker can avoid being involved in data movement since this will consume the bulk of the bandwidth. An adversary may try to convince the system that a particular node was unavailable or a new node joined (falsely). However our model allows the adversary no way of accomplishing the former; the latter case it will involve acknowledged data transfers which the adversary can not correctly acknowledge. Other rebalancing involve the adversary node itself, requiring it to be involved in the data movement.

Any distributed hash table must provide a mechanism to deal with this problem, regardless of whether there are malicious nodes present. Previously it was shown that in some file sharing systems, peers join and leave the system very rapidly [4]. The amount of data stored and the rate of replication at each node must be kept at levels that allow for timely replication without causing network overload, even when regular nodes join and leave the network.

Unsolicited messages Sometime a malicious node is able to create a situation where it can send an unsolicited response to a query. For example, consider a lookup process where querier Q referred by node N to node A. Node N knows that Q's next contact A, presumably with a follow-up to the query just processed by N. Thus N can attempt to forge a message from A to Q with incorrect results.

Employing standard authentication techniques such as digital signatures or message authentication code would be the best defense against this. Since, digital signatures are expensively currently and MAC's require shared keys. A more reasonable defense might include a random nonce with each query to ensure that the response is accurate.

5. Conclusion

This paper categorized and presents with examples the basic attacks which a peer-to-peer hash lookup systems must be aware of. Here it discusses all the details of such attacks as applied to some specific systems, and also suggests defenses in many cases. It then accumulates these defenses into a set of general design principles: (a) Allow the querier node to observe lookup progress, (b) Define verifiable system invariants by a node, (c) Assign keys to nodes in a verifiable way, (d) Server selection in routing may be abused, (e) Cross-check routing tables using random queries, (f) Avoid responsibility to a single point (nodes).

References

[1] Dabek, F., Kaashoek, M. F., Karger, D.,Morris, R., and Stoica, I. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM SOSP* (Banff, Canada, Oct. 2001), pp. 202–215.

[2] Freedman, M. J., Sit, E., Cates, J., and Morris,R. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the First InternationalWorkshop on Peer-to-Peer Systems* (Cambridge,MA, Mar. 2002).

[3] Fu, K., Kaashoek, M. F., and Mazi`eres, D.Fast and secure distributed read-only file system. In *Proceedings of the 4th USENIX Symposium on OperatingSystems Design and Implementation (OSDI)* (Oct. 2000), pp. 181–196.

[4] Krishnamurthy, B., Wang, J., and Xie, Y. Early measurements of a cluster-based architecture for P2P systems. In *Proceedings of the First ACMSIGCOMM Internet Measurement Workshop* (San Francisco, California, Nov. 2001), pp. 105–109.

[5] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. A scalable content addressable network. In *Proceedings of ACM SIGCOMM* (San Diego, California, Aug. 2001), pp. 161–172.

[6] Rowstron, A., and Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)* (Nov. 2001).

[7] Rowstron, A., and Druschel, P. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM SOSP* (Banff, Canada, Oct. 2001), pp. 188–201.

[8] Rowstron, A., Kermarrec, A.-M., Castro, M., and Druschel, P. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication: Third International COST264 Workshop* (Nov. 2001), J. Crowcroft and M. Hofmann, Eds., vol. 2233 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 30–43.

[9] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM* (San Diego, California, Aug. 2001),pp. 149–160.

[10] Zhao, B., Kubiatowicz, J., and Joseph, A. Tapestry: An infrastructure for fault-tolerant widearea location and routing. Tech. Rep. UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, Apr. 2001.