Detecting and Recovering from Overlay Routing Attacks in Peer-to-Peer Distributed Hash Tables

MS Thesis Defense
Keith Needels
March 20, 2009
Thesis Information

- Committee:
  - Chair: Professor Minseok Kwon
  - Reader: Professor Alan Kaminsky
  - Observer: Professor Warren R. Carithers

- Website:
  - http://www.csh.rit.edu/~keithn/thesis

- Publication:
  - ACM Symposium on Applied Computing, March 2009
Distributed Hash Tables (DHTs)

- Distributed hash tables (DHTs) provide a lookup mechanism for peer-to-peer networks.
- Unlike centralized server approaches, DHTs are fully distributed. Unlike flooding approaches, DHTs are scalable.
- Peers in the network (*nodes*) and keys for desired items are both mapped to *identifiers*. DHT protocols define a structure for a p2p overlay network and a protocol to follow for finding nodes responsible for storing information about keys that map to a range of the network’s identifier space.
Chord\textsuperscript{[1]}- A popular DHT

- Chord’s identifiers are 160 bit long integers. Nodes and keys are hashed to identifiers using SHA-1.
- The identifier of a node is the hash of its IP address.
- Each key’s value is stored on the first node whose identifier succeeds the key’s identifier.
  - Identifiers are arranged in a ring. The identifier that succeeds $2^{160}-1$ is 0.
- Keys may be hashed in different ways to map them to multiple nodes – replication.
- Chord is efficient and scalable!
  - Nodes must keep $O(\log n)$ routing entries.
  - Lookups take $O(\log n)$ hops.
Chord Node and Key Organization ($m = 10$)
Chord Lookups

- Each node has a reference to the node “half way” across the ring, “quarter way” across the ring, 1/8th of the way across the ring, and so on.
  - I have a reference to the nodes responsible for key (myID + 2^i) for i = 0 to m-1, where m is the identifier bit length (160 in Chord).

- Lookup requests are routed to the closest node I have a reference to whose identifier precedes the key.

- Routing tables in Chord are referred to as “finger tables.”
  - We will refer to each (myID + 2^i) value for i = 0 to m-1 as the finger pointer for table entry i.
Chord Finger Table \((m = 10)\)
Chord routing example \((m = 10)\)
Security Issues

- In order for Chord lookups to succeed, all nodes on the route to the destination must cooperate and follow the protocol correctly.
- Malicious nodes can easily drop, misroute, and modify routing requests.
- A relatively small percentage of compromised nodes can disrupt a large percentage of lookup requests.
Attacks Considered for this Thesis

- We will assume that a minority of nodes in the system are compromised. We are *not* defending against a Sybil attack.
  - There are other mechanisms for defending against massive amounts of nodes joining to launch a Sybil attack.\(^2\)\(^3\) We are considering the case where a relatively small subset of nodes are compromised.

- We also assume that nodes cannot choose their identifiers.
  - Chord calls for nodes to hash their IP addresses to find their identifiers. We can alternatively have a CA grant identifiers.\(^4\)\(^5\)\(^6\) Both mechanisms can be verified by other nodes.
Attacks Considered for this Thesis (continued)

Here are some of the things our attackers can do:

- Drop lookup requests.
- Misroute lookup requests, either while colluding or not colluding with other malicious nodes.
- Try to modify lookup requests.
- Be selective in which lookup requests they attack.

It only takes a small fraction of nodes to compromise a large fraction of lookups.

- In a 1,000 node system with an average hop count of 5, if 25% of nodes are compromised 76% of all lookups will be compromised.
Defense Overview

1. Use *iterative routing* instead of *recursive routing*. Recursive routing gives us no control over the routing process.
2. Validate each hop on the route to the destination using locally calculated statistics about the network.
3. When a hop fails validation, backtrack around the node that provided that hop.
Revised Finger Table

- In order to compute statistical data for hop validation, we need to store additional information in our finger table.
  - We will now store the identifiers of the predecessors and successors of our finger table nodes.

- New finger table row format:

| Index | Node Identifier and Reference | Node Predecessor Identifier and Reference | Node Successor Identifier and Reference |
Iterative Routing

- Iterative routing means the node performing the lookup queries each node on the path to the destination for the next node on the route.
- Instead of asking for the next hop to the destination, we will ask for a node’s entire finger table and find the next hop ourselves.
- We cache finger tables throughout the instance of a single lookup request so that we don’t have to request it again while backtracking.
Backtracking

- If a hop fails validation, we backtrack to the last node on the path and use its finger table to find the second closest preceding node to the destination.
  - The node that provided the “bad” hop will never be used again for this lookup.
  - If we backtrack to a node again, we use its third closest preceding node (and so on.)
- If a router has no more hops to give, we backtrack again and stop using that router for the rest of the lookup request.
Solid nodes are uncompromised, hollow nodes are compromised.
Bypassing Nodes

- If all of the possible next hop nodes provided by a routing node give hops that fail validation, but the routing node happens to have a reference to the destination node and that hop passes validation, then we skip right to the destination.

- This allows us to avoid what would otherwise be an impassable cluster of compromised nodes.

  - This also allows us to find the destination node if its predecessor is compromised.
Bypassing Illustrated
Hop Validation

- We use the additional data stored in our finger table to calculate the average distance between nodes in our Chord ring as well as the standard deviation.
  - More on this in a minute…

- If the distance between a finger table pointer and a finger table node is too many standard deviations over the calculated average distance between nodes in the system, that hop fails validation.
  - The number of standard deviations over the average that we consider acceptable is a configurable parameter. We refer to this as the standard deviation parameter.
Distance Samples

Source Node

Finger Table Nodes
Validation Illustrated

Solid nodes are uncompromised, hollow nodes are compromised.
Validation Illustrated

Solid nodes are uncompromised, hollow nodes are compromised.
Calculating Statistics

- Malicious nodes in our finger table can pollute our distance sample set with large values.
  - Malicious nodes cannot give samples that are too small. We can verify that the claimed successor/predecessor of a node is valid and is in the network.
- Large samples will cause us to overestimate the distance between nodes and lead to more false negatives during hop verification.
Distance Sample Pruning

- The distance between node IDs on a Chord ring (or any ring DHT) is exponentially distributed as long as identifiers are randomly distributed throughout the ring.[6]

- With an exponential distribution, the average of all samples is equal to the standard deviation of those samples.

- To prune, we continually remove the largest distance samples until the standard deviation of those samples is within range of some multiple of the average.
  - This multiple is a configurable parameter called the *pruning parameter*. 
Attack: Dropping routing requests

- With this type of attack, a node simply refuses to route our lookup request.
- This should be the easiest attack for us to recover from. We don’t need to rely on our malicious hop detection algorithm, only our modified backtracking routing algorithm.
  - We can test the backtracking algorithm independently of the hop detection algorithm. This will show how our system would perform with 100% accurate malicious hop detection.
- Experiment: 1000 nodes, varying percentage of malicious dropper nodes, infinite standard deviation parameter (malicious hop detection disabled).
Lookup Success Rate with Dropper Nodes (1000 Node System)

Hop verification disabled (Standard deviation parameter = infinity)
Average Hop Count with Dropper Nodes (1000 Node System)

Hop verification disabled (Standard deviation parameter = infinity)
Attack: Randomly routing lookup requests

- With this type of attack, a malicious node routes lookup requests to another random malicious node.
- Unlike with the previous attack, it now appears that our attacker is cooperating.
- Now we have to use the malicious hop detection algorithm, but detection should not be too difficult since most random hops will have much larger finger pointer to node distances than the locally computed average distance between nodes.
- Experiment: 1000 nodes, varying percentage of randomly routing nodes, standard deviation parameter $= 3.0$, pruning parameter $= 1.0$. 
Lookup Success Rate with Randomly Routing Nodes (1000 Node System)

Success - Modified
Deceived - Modified
Success - Default

Standard deviation parameter = 3.0, Pruning parameter = 1.0
Average Hop Count with Random Router Nodes (1000 Node System)

Percent of Nodes Randomly Misrouting Lookup Requests to Eachother

Standard deviation parameter = 3.0, Pruning parameter = 1.0
Attack: Colluding sub-ring

- This is an attack where malicious nodes route requests to the closest preceding malicious node in the network.
  - The attacking node maintains a finger table of only compromised nodes and provides that finger table when requested.
- This attack will be more difficult to detect than the random attack since the attacker is now doing the best that it can to appear to be cooperating.
- Experiment: 1,000 and 10,000 nodes, varying percentage of compromised nodes, standard deviation parameter = 1.75 and 3.0 with each network size, pruning parameter = 1.0.
Lookup Success Rate With Colluding Sub-ring Nodes (1,000 Nodes)

Successful Lookups (Percent)

Percent of Nodes Colluding

- Modified - SD_PARAM of 1.75
- Modified - SD_PARAM of 3.0
- Default

Pruning parameter = 1.0
Average Hop Count With Colluding Sub-ring Nodes (1000 Node System)

Pruning parameter = 1.0

- Modified - SD_PARAM of 1.75
- Modified - SD_PARAM of 3.0
- Default
## Colluding attack data points

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Malicious %</th>
<th>Modified Success %</th>
<th>Unmodified Success %</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000 nodes, 3.0 standard deviation parameter</td>
<td>14%</td>
<td>92%</td>
<td>56%</td>
<td>36%</td>
</tr>
<tr>
<td>10,000 nodes, 3.0 standard deviation parameter</td>
<td>14%</td>
<td>93.5%</td>
<td>43%</td>
<td>50.5%</td>
</tr>
<tr>
<td>1,000 nodes, 3.0 standard deviation parameter</td>
<td>26%</td>
<td>77.5%</td>
<td>44.5%</td>
<td>33%</td>
</tr>
<tr>
<td>10,000 nodes, 3.0 standard deviation parameter</td>
<td>26%</td>
<td>76%</td>
<td>38%</td>
<td>38%</td>
</tr>
</tbody>
</table>
Measuring Parameter Impact

- In order to obtain a better understanding of how the standard deviation parameter and pruning parameter affect the performance of the system, I ran experiments where only those parameters were modified.

- The results are shown over the next few slides.
  - But first, a review of the two parameters…
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Effect of Modifying</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Deviation Parameter</td>
<td>Controls how many standard deviations above the average node distance a finger pointer to node distance is allowed to be during hop validation.</td>
<td>A higher value will decrease false positives and increase likelihood of finding the correct destination. However, a higher value also increases false negatives and increases the average hop count.</td>
</tr>
<tr>
<td>Pruning Parameter</td>
<td>Controls how many multiples of the average node distance the standard deviation of node distances is allowed to be while pruning.</td>
<td>Too low and the calculated average node distance will be too low, resulting in more false negatives and failed lookups. Too high and the calculated average will be too high, resulting in more false positives and incorrect lookups.</td>
</tr>
</tbody>
</table>
Effect of Standard Deviation on Lookup Success (1000 nodes, 25% Colluding)

Successful lookups (Percent)

Standard Deviation Parameter

Successful lookups
Deceived lookups
Failed lookups

Pruning parameter = 1.0
Effect of Standard Deviation on Hop Count (1000 nodes, 25% colluding)

Average Hop Count

Standard Deviation Parameter

Pruning parameter = 1.0
Effect of Pruning on Lookup Success (1000 nodes, 25% Colluding)

Standard deviation parameter = 1.75
Conclusion

- In the face of naïve attacks, the modified algorithm works well.
- In the face of a sophisticated attack, the modified algorithm’s success rate drops but is still significantly better than the unmodified Chord algorithm.
- Our algorithm does not require a great deal of extra information and does not fundamentally change the underlying overlay structure.
  - This means our solution should easily combine with other Chord enhancements.
Questions?
References (presentation only)


SUPPLEMENTAL SLIDES
Data for 100 nodes – Not used in the paper

EXTRA DATA
Effect of Standard Deviation on Hop Count - 100 nodes, 25% colluding
Effect of Standard Deviation on False Positive/Negative Rates - 100 nodes, 25% colluding

Fraction of All Hop Tests

Standard Deviation Parameter

False Positive Rate

False Negative Rate
Effect of Pruning on Lookup Success - 100 nodes, 25% Colluding
Effect of Pruning Parameter on Estimated Network Size - 100 nodes, 25% Colluding
Effect of Standard Deviation on Lookup Success - 100 nodes, 25% Colluding

- Successful lookups
- Incorrect lookups
- Failed lookups
DHT OVERVIEW
Peer-to-Peer (P2P) Systems

- Distributed systems without central servers.
- Every participating computer (node) is both a client and a server.
- Nodes are connected to each other in an overlay network.
- Can be used for more than file sharing!
  - Distributed file systems
  - DNS (and any other naming system)
  - Newsgroups
  - Web Caching
Peer-to-Peer Systems

- The key function of a peer-to-peer system is to find nodes who have a resource that we are looking for.

- “Bad” approaches:
  - Napster – Not fully P2P: Central server
  - Gnutella – Slow, not scalable, lookup failures

- The solution?
  - Distributed Hash Tables! (DHTs)
Distributed Hash Tables (DHTs)

- Traditional hash tables map key values to array indices in constant time.
- Distributed hash tables map key values to nodes.
  - We would like to only keep track of a small subset of the nodes in the DHT, so we’ll have to route lookups through other nodes.
  - We want to find the node responsible for a key with as few hops through other nodes as possible.
Traditional Hash Table

1. HASH(Kaminsky)
2. Return 1400
3. Retrieve OfficeArray[1400]

OfficeArray

0
1
2
3
...
568
569
570
571
572
573
...
1395
1396
1397
1398
1399
1400
1401
...

Kwon 70-3547
Homan 70-3519
Raj 70-3619
Kaminsky 70-3635
Distributed Hash Table

1. HASH(Kaminsky)
2. Return 1400

4. lookup(1400)
5. Return 143.21.63.157

7. RETRIEVE(Kaminsky)

1257 - 1394
1395 - 129
130 - 295
1112 - 1256
296 - 503
504 - 667
801 - 1111
668 - 800

Hash Function
Security Issues with DHTs

- Since I can only maintain information about some subset of the nodes in the system, I have to rely on others to route my lookup requests to their destination.
- What if one of the nodes on the route is an attacker?
  - They could drop my request
  - They could return the wrong node or a malicious node
  - They could send bad routing information to other nodes
  - ...and much more.
Example Attack

1. HASH(Kaminsky)
2. Return 1400
3. lookup(1400)
4. Return 66,66,66,66
5. RETRIEVE(Kaminsky)

“Kaminsky” is the **key**

1400 is the **identifier**
ALGORITHMS
Revised Closest Preceding Node Algorithm

```python
n.next_hop(id, index, nodeid, fingertable)
    uc = 0
    if id in range(nodeid, fingertable[1]):
        if index == 1:
            return fingertable[1]
        else:
            return null
    bypassnode = null
    for i = m down to 1:
        if id in range(fingertable[i].predecessor, fingertable[i]):
            bypassnode = fingertable[i]
        if fingertable[i] in range(n, id):
            if (i == m or fingertable[i] != fingertable[i+i]):
                uc = uc + 1
                if (uc == index):
                    return fingertable[i]
    return bypassnode
```
Revised Find Successor Algorithm

\[
\text{\texttt{n.find_successor(id, hoplim):}}
\]

\[
\begin{align*}
\text{\texttt{routerStates = new STACK of <id, fingertable, index> tuples}} \\
\text{\texttt{blackList = new LIST of nodes}} \\
\text{\texttt{attempts = 0}} \\
\text{\texttt{routerStates.push(n.identifier, n.fingertable, 0);}} \\
\text{\texttt{while !routers.isEmpty() and attempts < hoplim:}} \\
\text{\texttt{curRouterState = routerStates.pop()}} \\
\text{\texttt{curRouterState.index++}} \\
\text{\texttt{nextHop = n.next_hop_forward_bypass(id, curRouterState.index, curRouterState.id, curRouterState.fingertable);}} \\
\text{\texttt{if (nextHop != null and verify_hop(curRouterState.id, nextHop.id) and !routerStates.contains(nextHop)) or curRouterState.id = n.id:}} \\
\text{\texttt{if id in range (n.id, nextHop.id):}} \\
\text{\texttt{return nextHop}} \\
\text{\texttt{else if blacklist.contains(nextHop.id):}} \\
\text{\texttt{routerStates.push(curState)}} \\
\text{\texttt{else:}} \\
\text{\texttt{routerStates.push(curRouterState)}} \\
\text{\texttt{routerStates.push(<nextHop.id, nextHop.fingerTable, 0>}} \\
\text{\texttt{attempts++}} \\
\text{\texttt{else:}} \\
\text{\texttt{blackList.add(curRouterState.node.id)}} \\
\text{\texttt{return null}}
\end{align*}
\]
Hop Verification Algorithm

\[ n\text{.verify\_hop}(first\text{Node}\_Id, second\text{Node}\_Id, index\text{Used}) \]

\[
\text{fingerPointer} = (first\text{Node}\_Id + \text{pow}(2, index\text{Used})) \mod 2^m
\]

\[
distance = second\text{Node}\_Id - \text{fingerPointer} \mod 2^m
\]

\[
\text{acceptableDistance} = AVG\_DISTANCE + (sd\_mod * STD\_DISTANCE)
\]

\[
\text{if} \ (distance > \text{acceptableDistance}) : \\
\quad \text{return false}
\]

\[
\text{else:} \\
\quad \text{return true}
\]
Calculate Statistics Algorithm

\[
n.n.\text{calculate}\_\text{statistics}(\text{distanceSamples} \text{ (LIST of } <\text{BigInt}>)\text{, } \text{pruningParameter})
\]
\[
\text{done} = \text{false}
\]
\[
\text{average} = \text{stdeviation} = 0
\]
\[
\text{distanceSamples}.\text{sortAscending}()
\]
\[
\text{while} (\text{!done}):
\]
\[
\text{average} = \text{AVG}(\text{distanceSamples});
\]
\[
\text{stdeviation} = \text{STDEV}(\text{distanceSamples});
\]
\[
\text{if} \ \text{stdeviation} > \text{average} \ast \text{pruningParameter}:
\]
\[
\qquad \text{distanceSamples}.\text{remove}(\text{distanceSamples}.\text{size()} - 1)
\]
\[
\text{else}:
\]
\[
\qquad \text{done} = \text{true}
\]
\[
\text{return} (\text{average}, \text{stdeviation})
\]
JOIN PROTOCOL
Joining the Network

- This is the one scenario where trust must exist. We must, out of band, retrieve a set of uncompromised nodes.
- To join, we ask each node in the bootstrap list to perform a lookup of each of our finger pointer identifiers ($nodeid+2^i$ for $i$ from 0 to $m-1$).
  - For each entry, we accept the node that was closest to our finger table entry.
MISCELLANEOUS
Security Issues (continued)

- If the average hop count is $h$ and the fraction of malicious nodes is $f$, then the probability of a route containing no malicious nodes is $(1-f)^h$.

- With Chord, the average hop count of an $n$ node network is approximately $\frac{1}{2} \log_2 n$.
  - With a 1,000 node network, this means the average hop count will be around 5.

- If 25% of the nodes in a 1,000 node Chord network are compromised, the probability of a routing request avoiding any malicious node is $0.75^5$, or 24%.
  - An attacker only needs to control 25% of the nodes in a 1,000 node system to disrupt 76% of lookups!
Dropped routing request result

- The system performs well in the presence of malicious dropper nodes.
- When half of all nodes are compromised, the unmodified Chord algorithm finds the correct node 6% of the time, while our modified algorithm finds the correct node 94.7% of the time.
- The average hop count increases slowly at first, but then more quickly once over 25% of nodes are compromised:
  - 0% compromised: 4.84 hops
  - 25% compromised: 9.15 hops
  - 50% compromised: 20.50 hops
Random routing attack results

- With 50% of nodes compromised, the modified algorithm finds the correct node 91% of the time while the unmodified algorithm still only finds it 6% of the time.
- Note the hop count differences between the dropper attack and random router attack:
  - With no nodes compromised, the modified algorithm adds about one hop to every lookup due to false positives with the malicious hop detection algorithm.
  - With 50% of nodes compromised, our average hop count is 3 hops less than with the malicious dropper nodes. This is due to false negatives that result in us reaching the wrong destination and not backtracking to find the correct one.
Colluding sub-ring attack results

- With less than 25% of nodes compromised, the modified algorithm significantly outperforms the unmodified algorithm. This difference is not as great when more than 25% are compromised.
- The larger the network, the greater the margin is that the modified algorithm outperforms the default protocol.
- Although the modified algorithm outperforms the default protocol for this type of attack, it does not perform as well as it does for the previous two types of attack.
Parameter Test Results

- No big surprises in these test results.
- The optimal pruning parameter value is around 1.0. This makes sense.
  - With a random node distribution, the average distance between nodes should equal the standard deviation.
Effect of Standard Deviation on False Positive/Negative Rates (1000 nodes, 25% colluding)

Pruning parameter = 1.0
Effect of Pruning Parameter on Estimated Network Size (1000 nodes, 25% Colluding)

Standard deviation parameter = 1.75
Successful Lookups (Percent)

Lookup Success Rate with Colluding Sub-ring Nodes (10,000 Nodes)

Percent of Nodes Colluding

Pruning parameter = 1.0
Average Hop Count With Colluding Sub-ring Nodes (10,000 Node System)

Average Hop Count

Percent of Nodes Colluding

Modified - SD_PARAM of 1.75
Modified - SD_PARAM of 3.00
Default

Pruning parameter = 1.0