Stealth Distributed Hash Table: A Robust and Flexible Super-Peer DHT

A. Brampton, A. MacQuire, I. Rai, N. Race, L. Mathy

In Proceedings of the 2nd Conference on Future Networking Technologies (CoNEXT'06)

LANCASTER UNIVERSITY
1. A. Brampton, A. MacQuire, I. Rai, N. Race, and Laurent Mathy. *Stealth Distributed Hash Table: Unleashing the real potential of peer-to-peer*. In Proc. of CoNEXT’05 (Student Workshop Session).

2. This paper.


Introduction on DHT

• A common design approach for Distributed Hash Tables (DHTs) simply considers interconnecting *autonomous* and *homogeneous* nodes on the same overlay.

• These assumptions are unrealistic for any practical large scale network:
  - natural inequality in the capabilities
  - behavior of peers → *churn*

• A classical approach to improving this situation is to leverage the natural heterogeneity in the system by using superpeers.

• However, most DHT-based super-peer proposals rely on multiple overlays and suffer from a rather static binding between peers and super-peers.
The Stealth DHT proposal

• The *Stealth Distributed Hash Table* (*Stealth DHT*) differs from traditional DHTs mainly in that it makes a subset of nodes effectively *invisible* to the routing and forwarding decisions on the network.

• A Stealth DHT is composed of two kinds of node:
  - *Stealth nodes*
  - *Service nodes*

• Service nodes are highly stable and capable machines, while stealth nodes are heterogeneous, autonomous devices owned by end-users.

• Salient features of a Stealth DHT:
  - it maintains a single overlay
  - there exists no single point of failure
  - any source node uses the original routing decision process
DHT Overview

- DHTs serve as an object location service that can be used as a substrate for multiple large-scale distributed applications
- Many DHT algorithms have a similar structure:
  - Each node on the network has a unique identifier (ID)
  - The address space is dynamically partitioned into regions
  - Each region is then assigned to a single node
  - Each key in the DHT is generated by applying a hash function to the object it represents, producing an identifier that falls within a specific region of the address space
  - The DHT algorithm ensures that the key is held by the node responsible for that region
Service Node Join

- Knowing one established node on the network (identified as *bootstrap* node), the new node must first initialize its routing table.
- The join procedure in traditional DHTs has two phases:
  - a *state gathering* phase
  - an *announcement* phase
- A service node joins the DHT in conformance with the method prescribed by the original DHT.
Stealth Node Join

- A stealth node joins the Stealth DHT by only completing the state gathering phase. The node used as bootstrap node should be a service node.

- Implementing no announcement phase implies that:
  - a stealth node never appears in any routing table
  - no service node ever learns to route through a stealth node

- Stealth nodes are capable of injecting messages into the DHT, but are never used to relay any message.

- Benefits:
  - several stealth nodes may choose the same node ID without collisions being detected
  - the lack of announcement messages cuts the overhead of joining stealth nodes significantly
Stealth Routing State

- The service nodes that handle the join message for a stealth node should only provide such node with routing information contained in the first row of their routing table.
- When a service node leaves the network, some stealth nodes will have obsolete information for that node in their routing tables.
- Stealth nodes never receive announcement messages from newly arrived service nodes, a method is therefore required for maintaining routing state:
  - rejoining
  - periodic polling
  - piggybacking
Simulations

• In order to evaluate the Stealth DHT proposal, a discrete-event packet-level simulator for Pastry and Stealth DHTs has been developed

• Evaluations have been performed considering the following aspects:
  - Join Performance
  - Storage and Retrieval
  - Load-Balancing
Join Performance

- They compared the overhead of join operations between Pastry and a Stealth DHT by first measuring the number of messages generated during the construction of the DHT.
- Another metric they considered is the *join latency*: the time elapsed between a node sending its initial Join message and it receiving a JoinFinished message.
Storage and Retrieval

- They measured the **lookup latency**: the time elapsed between a node performing a get for a specific key in the DHT and it receiving a reply.
- Then they measured the **misses under churn** with and without replication.

![Graphs showing the decrease in lookup latency and misses under churn over DHT network size.](image-url)
Load-Balancing

- The comparison between Pastry and Stealth DHT has been performed measuring the CDF (Cumulative Distribution Function) of received messages per node both with and without churn.

![](image)
Implementation

• They created and deployed an implementation of a Pastry-based Stealth DHT onto PlanetLab, and compared the performance of Stealth DHT to that of Pastry while running on a real-world platform.

• They measured the number of messages generated on average when a single node (service or stealth) joins the network as a function of network size.
Applications of Stealth DHT

• Any application that can make use of a traditional DHT could also be implemented on top of a Stealth DHT
• A good example is a mobile environment:
  - nodes are particularly likely to cause churn
  - nodes benefit in joining the network quickly and efficiently
Thanks for Your Attention