Peer To Peer Systems & Security

Anonymous P2P VoIP System

Interims Report

DHT - Subproject

Team Alpha

Elias Hazboun

Nedko Nedko

07.06.2015

<u>Protocol structure:</u>

For our DHT subproject, we have decided to make it based on Kademlia and amend changes to the original protocol as we see fit for our specific use case. We have chosen Kademlia because it offers built-in replication, performs well under churn, has resistance against DoS attacks, has low overhead on the network and finally because it is relatively easy to implement and maintain.

The process architecture is implemented in two threads: one main thread that uses an event loop for processing requests and replies and storing data, the other is responsible for maintaining the k-bucket neighbors and data expiration.

All communication in the module protocol will be in UDP and the program will accept as an argument the IP (IPv4 or IPv6) and port number on which it will accept communication. All messages shall be encrypted. The protocol will exchange information in Binary JSON (BSON) due to its relative easiness to read and write (both for humans and machines) and is easily processed in the code.

<u>Message Types:</u>

Our module protocol has 10 message types:

1. PING: Used to check if a DHT peer is still alive.

{ “TYPE”:”PING”,
“MID”: *random 160 bit message identifier,*
“SID”: *256 bit sender ID,*
“RID”: *256 bit receiver ID* }

2. PONG: Response to PING messages.

{ “TYPE”:”PONG”,
“MID”: *random 160 bit message identifier,*
“SID”: *256 bit sender ID,*
“RID”: *256 bit receiver ID* }

3. STORE: Used to store key value pairs at a peer.

{ “TYPE”:”STORE”,
“MID”: *random 160 bit message identifier,*
“SID”: *256 bit sender ID,*
“RID”: *256 bit receiver ID,*
“Key”: *256 bits,*
“TTL”: *integer (seconds),*
“Value”: *content to be stored* }

4. STORE_REPLY: Used as a reply for the STORE request.

{ “TYPE”:”STORE_REPLY”,
 “MID”: *random 160 bit message identifier,*
 “SID”: *256 bit sender ID,*
 “RID”: *256 bit receiver ID,*
 “Status”: *integer* }

Status indicates the success or failure of the store request and in case of failure provides the associated error code.

5. FIND_NODE: Used to find the k closest nodes (as known by the receiver) to a given key.

{ “TYPE”:”FIND_NODE”,
 “MID”: *random 160 bit message identifier,*
 “SID”: *256 bit sender ID,*
 “RID”: *256 bit receiver ID,*
 “KX_INFO”: *boolean (request KX_INFO in reply),*
 “Key”: *256 bits* }

To accommodate the DHT TRACE request, we added the KX_INFO field. When the KX_INFO is set to true, the responder shall include in the reply its own IPv4, IPv6 and port triplet on which it expects KX connections.

6. FIND_NODE_REPLY: Used as reply for the FIND_NODE request.

{ “TYPE”:”FIND_NODE_REPLY”,
 “MID”: *random 160 bit message identifier,*
 “SID”: *256 bit sender ID,*
 “RID”: *256 bit receiver ID,*
 “KX_INFO_REPLY”:[*IPv4, IPV6, Port*],
 “Nodes”:[[*IPv4,IPv6,Port ID*], [*IPv4,IPv6,Port ID*], [*IPv4,IPv6,Port ID*]..] }

KX_INFO_REPLY field shall be populated with the IPv4, IPv6 and port triplet on which the peer expects KX connections if KX_INFO field was set to true in the FIND_NODE request.

7. FIND_VALUE: Used to find the value(s) for a given key, if it does not exist it functions as FIND_NODE request and returns the k closest nodes (as known by the receiver).

{ “TYPE”:”FIND_VALUE”,
 “MID”: *random 160 bit message identifier,*
 “SID”: *256 bit sender ID,*
 “RID”: *256 bit receiver ID,*
 “Key”: *256 bits* }

8. FIND_VALUE_REPLY: Used as reply for the FIND_VALUE request.

{ “TYPE”:”FIND_VALUE_REPLY”,
  “MID”: *random 160 bit message identifier,*
  “SID”: *256 bit sender ID,*
  “RID”: *256 bit receiver ID,*
  “Nodes”:[[*IPv4,IPv6,Port ID*], [*IPv4,IPv6,Port ID*], [*IPv4,IPv6,Port ID*]..],
  “Values”:[[*value*],[*value*]..] }

A key could be associated with multiple values. Either Nodes or Values fields shall be populated; if a node does not have the corresponding key stored locally, it will return the k closest nodes to that key.

9. VERIFY: Used to verify a node contacted for the first time by asking proof of work.

{ “TYPE”:”VERIFY”,
  “MID”: *random 160 bit message identifier,*
  “SID”: *256 bit sender ID,*
  “RID”: *256 bit receiver ID,*
  “Challenge”: *large random integer* }

This message is used for authentication and making sure that a legitimate peer is running on that host. If no VERIFY_REPLY message is received within a time period or the challenge is not successfully done, the host is considered not running a proper DHT instance.

10. VERIFY_REPLY: Used reply to VERIFY request.

{ “TYPE”:”VERIFY_REPLY”,
  “MID”: *random 160 bit message identifier,*
  “SID”: *256 bit sender ID,*
  “RID”: *256 bit receiver ID,*
  “Challenge_REPLY”: *[Integer1, Integer2]* }

To succeed the challenge, the peer must compute within a time limit a pair of values x, z such that the concatenation $x \mid y \mid z$, when run through a secure hash function, yields a value whose least significant n bits are all zero.

Some notes regarding the message types:

- MID is a quasi-random message ID generated for each request to make it somewhat unique, each client should include the same MID in its response, a peer receiving a reply with an unknown MID must discard the message and mark the originating peer as malicious.
- SID and RID are the SHA256 hashes of the public key of the sender and receiver peers respectively.

- SID must be the ID of the originator of the message, any peer found to be not using its own ID must be marked as malicious.
- RID must be verified upon receipt to be equal to the peer's own ID, otherwise message is discarded.
- Messages must be sent with source port equal to the port on which a peer is listening.
- For each sent request there shall be a reply. If a reply is not received within a given time period, the request is sent again for a maximum of 4 additional times after which if still no replies are received the peer is considered down and removed from the k-buckets.
- The same logic from the previous point shall be used for any reply that is received corrupted, but for a maximum of 2 additional times after which the peer will be marked as malicious also.
- Any peer marked as malicious shall be removed from the k-buckets and any future communication with it is refused.

For data corruption of the values, this could be easily detected, since the values are signed by the public key of a peer, and if such case is encountered the data is discarded and the peer is considered malicious. For other exceptions, we shall adhere to the original kademlia protocol specification whenever possible.