Peer To Peer Systems & Security

Anonymous P2P VoIP System

<u>Final Report</u>

DHT - Subproject

Team Alpha

Elias Hazboun

Nedko Nedko

06.09.2015

In this report, we present the cumulative outcome of our effort during the semester to create the DHT layer of the Anonymous VoIP project undertaken by the class. We will provide documentation of the software that we have delivered as well as future work and enumeration of any changes to the original protocol specification. We will also cover some project organizational matters.

Project Documentation

1. Requirements and Dependencies:
    a. Any Linux distribution
    b. Python 2.x
    c. Binary json for python
    d. Bitstring module for python
2. Installation
    a. To install binary json:
        i. "apt-get install python-bson"
    b. To install Bitstring module:
        i. "wget https://pypi.python.org/packages/source/b/bitstring/bitstring-3.1.3.zip"
        ii. Extract the file and change directory to the extracted folder
        iii. "python setup.py install"
    c. To setup and run the software:
        i. Edit the configuration settings in the "config/dht.conf" file
        ii. "python init.py" in the folder "source/modules/"
3. Known issues:
    a. The software is lacking some functionality in the "node-lookup" routine of Kademlia, the missing lines of code are currently written in pseudo code, but due to time constraints in testing, it was not written in actual lines of code.
    b. The software main components and routines were tested separately (Storage, buckets, protocol sending/receiving, etc...) but wholesome testing of the software was not done extensively due to point 3.a
    c. Some TODO comments in the code are erroneous, that is, their tasks were done in reality but the TODO comments were not removed by mistake.

Protocol Description/Changes

The protocol we implemented for the kademlia interactions was almost entirely done as described in the interim report. The following is a copy of the message specification we submitted in that report.

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.

Two notes about the specification above:

- Added “IP” and “port” fields in the ping and store messages that indicate where the originating peer is expecting/listening for kademlia messages. The receiving peer of these messages shall add sender peer to the buckets and assign the IP/port tuple to that peer.
- The verify/verify reply messages were implemented but are not currently used in this version of the project due to time constraints.

Future Work

Enhancements and features we want to implement in the future include:

- Finishing the “node-lookup” routine by rewriting the pseudo-code into actual code
- System testing, i.e. running a full-blown network of instances of the project on Mininet.
- Further testing for the effects of failed/malicious nodes and making sure the system can withstand such nodes.
- Deleting old erroneous comments in the code and some refactoring of code.
- Using the verify/store verify functions in the code.

Work Division

Work was divided as follows:

Software: Inter-module (DHT API) routines, inter-thread communication, socket programming and system configuration by Nedko. DHT data storage, Kademlia protocol, k-buckets system and maintenance by Elias.

Reports: Discussion was done together, writing was split Elias 70-30 Nedko.

Effort

Elias: 66 Hours

Nedko: 72 Hours