WESTERN UNIVERSITY - CANADA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SE 3314B – COMPUTER NETWORKS APPLICATIONS

Assignment 2: A DHT p2p network

Due Date: March 26th, 2024, at 11:55 pm

This assignment is to be done **individually**. Cheating is prohibited in this assignment, therefore do not to show your code to any other student, do not look at any other student's code, and do not put your code in any public domain. However, getting help is easy and available, just ask the instructor or the TAs. At the same time, you are encouraged to discuss your problems (if any) with other students in general terms, but **the actual program design and code must be your own**.

Please note that the consequences of cheating are much, much worse than getting a low mark for that particular item of work. The very best case is that you get a zero for the whole thing. Keep in mind that turning in your own work would have at least gotten you a few marks, better than a zero.

Code submissions will be graded manually, and the code similarity check will be made using the similarity-detection software system.

1 Objectives

The majority of socket programs, including GetImage/imageDB of assignment 1, follow the client- server paradigm, where a server waits on a well-known port for clients' connections. In this assignment, you'll explore the distributed hash table (DHT)-based peer-to-peer programming and implement a simplified Kademlia-like network. If you didn't manage to get your code to work in assignment1, no problem you still have the chance to practice socket-programming by exploring and studying the given sample solution for assignment 1. It is also expected, but not necessarily, to re-use some of the methods that were built for assignment1.

A DHT peer is basically both a server and a client. It accepts connections from other peers and connects to one or more peers. Before you continue working on this assignment, you must refresh your knowledge, on how DHT and Kademlia networks are built and work, by reading Unit 2 lecture slides 35-38 and 66-83.

2 Introduction

The main program you will develop is called KADpeer, it takes 5 arguments on the command line, two are mandatory (-n <peerName>) and the other three are optional :

```
> node KADpeer -n <peerNAme> [-p <peerIP>:<port>]
```

You must provide the -n option to assign a name to the peer instance. The other arguments are optional in certain context. If the peer is run without the -p option, it forms a new DHT Kademlia network with itself being the only peer overseeing the whole identifier space. The -p option specifies the target of the peer's message when joining an existing network. A peer's position in this network may not end up being adjacent to the target peer. A peer's ID (the default ID mode) is based on a 4 bytes shake256 hash of its IPv4 peerIP and port number port. Therefore, we assume 32-bit identifiers, which we will express in hex base (8 character address).

To bootstrap the Kademlia network, we first start a peer by itself. Every time a peer runs, it prints its IP address and the port number it is listening on, in addition to its ID. When a peer is run with the peerIP:port of another peer N as its command line argument, the new peer starts as a server listening on a random, ephemeral port, and at the same time, it tries to join the peer N by creating a socket and connecting to the peer N.

A peer that receives a join request will accept the peer if and only if its peer table is not full. Whether a join request is accepted or not, the peer sends back to the requesting peer the Ipaddress:port of a peer that is saved in its peer table (if the table is not empty). This will help the newly joined peer find more peers to join.

In this assignment, a custom Peer-to-Peer Transport Protocol (kadPTP) for peer communication will be built and used. The subsequent sections describe the structure and mechanism of this protocol.

2.1. Assumptions

To make the assignment tasks more manageable, we make the following simplifying assumptions:

- The peer information is an object holding the values of the peer's name, peer IPv4 address, the peer port number, and the peer ID.
- No peer departure: once a peer joins the network, it doesn't depart until you take down the whole network. This means that, you don't need to clean up after a peer departure, only be sure that you can take down the network without peer crashing.
- Peer join process does not fail. To assume otherwise would require a bit more complicated join protocol.
- No concurrent joins. Peers are added one at a time. Consequently, until a peer has completed its join process, it will interpret receiving a join packet from another peer as an error.
- Every time a peer needs to send a message, it opens a new connection to the target peer. The sender immediately closes the connection once the message is sent. So, there is no permanently opened connections. The only exception to this single message per connection rule is when performing on demand correction of the network inconsistency due to peer addition, as explained below.
- Each Kademlia peer keeps track of its known contacts in a routing table consists of up to 160 k-buckets. A k-bucket is a list of k contacts sorted by most recently inserted. In this assignment will set the value of k to be only 1

3 Task 1: Server Side

Your first task is to implement the server side of the KADpeer program. If KADpeer is launched without the -p options on the command line, it initializes a TCP socket and obtains an ephemeral port number from the kernel which it prints out to the console. It will also start a timer to tick every 10 milliseconds, that is initialized by a random value from 1 to 999 and keep incremented by 1 every tick along the lifetime of the KADpeer execution.

If a new peer (the connecting peer) is trying to connect to the KADpeer (the receiver peer), The receiver will accept the connection, display it and send back a welcome message packet (see Figure 1) with 'Message Type' field set to 1, (1 means Welcome) using receiver's handleClientJoining(sock) (given in assignment1). This message will include information about all the peers known by the receiver peer. Then the receiver will store the connecting peer's information into its DHT table (initially empty) using the pushBucket() method (described below), The stored information (commonPrefix, IP address, port number, and peer ID) will be placed into the appropriate k-bucket of its DHT table. Note that, the DHT table has 32 k-buckets that are numbered from 0 to 31. The appropriate k-bucket is the ith k-bucket in the DHT table, where

i is the number of the leftmost bits shared between the receiver (the owner of the DHT table) and sender IDs. The receiver peer should then print its DHT table. (The socket may remain open to handle a hello message from the same peer but should be closed right after.).The figure below shows the kadPTP message packet structure.

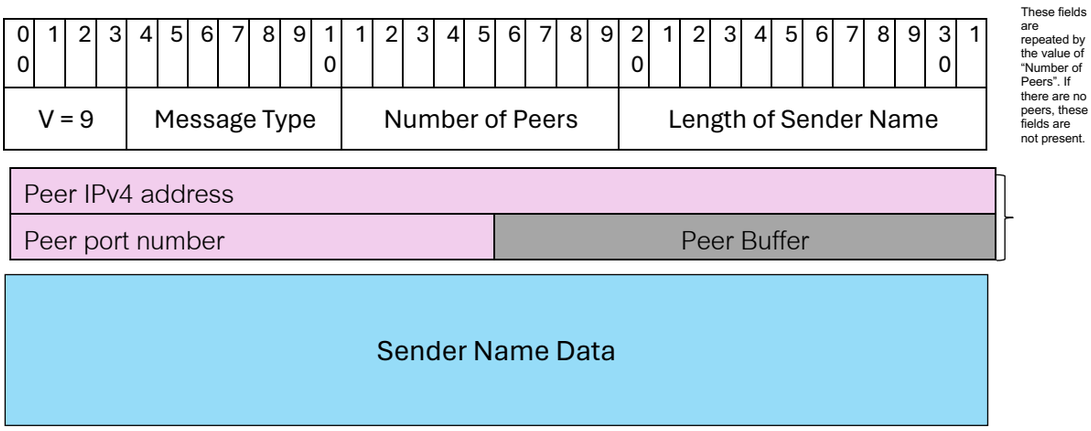

Figure 1 KadPTP message packet format

Here are the descriptions of the kadPTP protocol message packet fields:

- V) is a 4-bit ITP version field. You must set this to 9. The bits order should be as follows:
- The message type is a 7 bit field, the value 1 means 'Welcome'. Other types will be described and used later.
- Number of peers is a 9-bit field which holds the number of peers included in the message packet. If the peer's DHT table is empty, the 'Number of peers' field is thus set to 0. If the peer's DHT table is not empty, the 'Number of peers' field is set to number of peers stored in the DHT table. Note that, the message packet is sent before the new accepted peer is stored into the peer table.
- The length of Sender Name is a 12-bit field to hold the size of the sender's name in bytes (note that, sender name is not the sender ID)
- The 'length of Sender Name' field is followed by a list of peer table information. Each entry in this list consists of a 4-byte Peer IP address and 2-byte Peer port number and a 2-bytes buffer.
- The Sender name field comes last and holds the message sender name, the size of this field is dictated by the Sender name length field above.

3.1. pushBucket() method

This method takes two arguments: a DHT table, T, and a peer's information, P. The method adds P into the appropriate k-bucket in T as follows:

• Examine the bits value of both P and the peer owning T, say P′, to determine the maximum number of the leftmost bits shared between P and P′, say n.

• If the nth k-bucket in T is empty insert P into the nth k-bucket. A message should be displayed to indicate the addition.

• If the nth k-bucket is full, say it has the peer N, then do the following :

- Print which bucket is full.
- Determine which of the peers P and N is closer to P′. Remember Kademlia uses the exclusive OR operator to find the distance between two peers. Please review Unit 2. Display a message saying which of the 2 peers was added/kept and why.

3.2. Peer ID generation

In this assignment we will use the cryptographic hash function shake 256 to generate the IDs of the peers joining the DHT Kademlia network. The hash takes any string value and produces a fixed length output used to uniquely identify each peer in the network. For our implementation, we want this output to be of 4 bytes. The body of the method getPeerID is given in the Singleton.js file in the sample solution of assignment 1, you should use the getPeerID method whenever you need to generate an ID for a given IP and port number of a peer. The string given to the hash function should be consistent and use both the ip address and port number. Other useful functions are also added into the Singleton.js file. Hint: Use the crypto package.

4 Task 2: Client Side

If the KADpeer (the current host) is run with the -p option, the user must provide a known peer IP address and port number to connect to, with the port number separated from the peer IP address by a colon. Keep in mind that, a peer joining the p2p network simply connects to another peer, without sending any messages, so using a different version number with the -v command line option will not affect the join process.

Note that Node.js would have assigned a random, ephemeral source port to the connected socket. Find out the assigned ephemeral source port number and store it along with the IPv4 address of the current host and the generated DHT peerID. Note also that, the KADpeer program will be managing for activities on both the socket connected to the known peer (the receiver) and the socket on which the peer program (the sender) is listening for connection from other peers.

The peer program (the sender peer) checks for activity on the connected peer's socket. If there's an incoming packet, it receives the message and examines its packet. It first checks the version number of the received packet. If it is not ‘9’, the message will be ignored. Assuming the version number checks out, the receipt of a packet carrying a list of all known peers by the receiver causes these list peer's information to be printed out. Then the sender peer will run the refreshBuckets() to update its DHT table, as described below. Once the DHT table is updated, the sender’s peer will send Hello packet to all peers in its DHT table using the sendHello() method (described below)..

4.1 Refresh Buckets() method

This method takes two arguments: a DHT table, T, and a list of peers’ information, P1, P2, …, Pn. The method will attempt to add the peers in this list one at a time, using pushBucket() method, into the appropriate k-bucket of T (if possible). The method should then print the current DHT table.

4.2 sendHello() method

This method takes a DHT table, T, as an argument. The method scans the k-buckets of T and sends a hello message packet to every peer P in T, one at a time. It does so to insert itself into this peer P and let the peer P refresh its k-buckets using the transmitted table T. The structure of the hello message packet is shown in Figure 1, with ‘Message Type’ field set to 2,

(2 means Hello). This message will include information about all peers known by the sender of the hello message. And then the receiver does the following:

1) Stores using the pushBucket() method, the sender's peer information in the appropriate k-bucket.

2) Invokes the refreshBuckets() method to update its DHT table with the received list of peers known by the sender.

3) it closes the connection with the sender peer.

5 Sample Execution Run

A video is provided on owl showing you a sample execution run, but additionally below is a running scenario as an example to show the execution of the program and to summarize the requirements of this assignment.

Either using your ide or a command line interface, you will launch the first peer for the network, we will name this peer server:

```
> node KADpeer -n server
```

It should print the following to the screen (with a different port and ID numbers), note that the:

```
This peer address is 127.0.0.1:4984 located at server [7cf2cf17]
```

Where server is the name given through the command line and [peerID] is the generated peerID.

Using an other terminal, run KADpeer using the following command (where address and port number correspond to what was printed above.

```
>node KADpeer -n peer2 -p 127.0.0.1:4984
```

It should print to the screen (with different port numbers and time stamp):

```
Connected to peer1:4984 at timestamp: 657

This peer is 127.0.0.1:60935 located at peer2 [6f507657]

Received Welcome Message from server 7cf2cf17 along with DHT
[]

Bucket P7 has no value, adding 7cf2cf17

Refresh k-Bucket operation is performed

My DHT:
[ P7, 127.0.0.1:4984, 7cf2cf17]

Hello packet has been sent.
```

Meanwhile, on the first window, you should see the following additional line printed to screen:

```
Connected from peer 127.0.0.1:60935

Bucket P7 has no value, adding 6f507657
```

```
My DHT:
[ P7, 127.0.0.1:60935, 6f507657]

Received Hello Message from peer2 6f507657 along with DHT
[127.0.0.1:4984, 7cf2cf17]

Bucket P7 is full, checking if we need to change the stored value
Current value is closest, no update needed

Refresh k-Bucket operation is performed

My DHT:
[ P7, 127.0.0.1:60935, 6f507657]
```

Lets add an other peer.

```
>node KADpeer -n peer3 -p 127.0.0.1:4984
```

You should receive the following :

```
Connected to peer1:4984 at timestamp: 595

This peer is 127.0.0.1:60938 located at peer3 [02a06281]

Received Welcome Message from server 7cf2cf17 along with DHT
[127.0.0.1:60935, 6f507657]

Bucket P5 has no value, adding 6f507657

Bucket P5 is full, checking if we need to change the stored value
Current value is closest, no update needed

Refresh k-Bucket operation is performed

My DHT:
[ P5, 127.0.0.1:60935, 6f507657]

Hello packet has been sent.
```

From the server peer you will see:

```
Connected from peer 127.0.0.1:60938

Bucket P5 has no value, adding 02a06281

My DHT:
[ P5, 127.0.0.1:60938, 02a06281]
[ P7, 127.0.0.1:60935, 6f507657]
```

```
Received Hello Message from peer3 02a06281 along with DHT
[127.0.0.1:60935, 6f507657]

Bucket P7 is full, checking if we need to change the stored value
Current value is closest, no update needed

Bucket P5 is full, checking if we need to change the stored value
Current value is closest, no update needed

Refresh k-Bucket operation is performed

My DHT:
[ P5, 127.0.0.1:60938, 02a06281]
 [ P7, 127.0.0.1:60935, 6f507657]
```

Since peer2 was also added to peer3's DHT, they also received the hello message:

```
Received Hello Message from peer3 02a06281 along with DHT
[127.0.0.1:60935, 6f507657]

Bucket P5 has no value, adding 02a06281

Refresh k-Bucket operation is performed

My DHT:
[ P5, 127.0.0.1:60938, 02a06281]
[ P7, 127.0.0.1:4984, 7cf2cf17]
```

Lastly lets add one more peer:

```
Connected to peer1:4984 at timestamp: 624

This peer is 127.0.0.1:60944 located at peer4 [7e4a630f]

Received Welcome Message from server 7cf2cf17 along with DHT
[127.0.0.1:60938, 02a06281]
[127.0.0.1:60935, 6f507657]

Bucket P5 has no value, adding 02a06281

Bucket P7 has no value, adding 6f507657

Bucket P14 has no value, adding 7cf2cf17

Refresh k-Bucket operation is performed

My DHT:
```

```
[ P5, 127.0.0.1:60938, 02a06281]
[ P7, 127.0.0.1:60935, 6f507657]
[ P14, 127.0.0.1:4984, 7cf2cf17]

Hello packet has been sent.
```

The server will show:

```
Bucket P14 has no value, adding 7e4a630f

My DHT:
[ P5, 127.0.0.1:60938, 02a06281]
[ P7, 127.0.0.1:60935, 6f507657]
[ P14, 127.0.0.1:60944, 7e4a630f]

Received Hello Message from peer4 7e4a630f along with DHT
[127.0.0.1:60938, 02a06281]
[127.0.0.1:60935, 6f507657]
[127.0.0.1:4984, 7cf2cf17]

Bucket P5 is full, checking if we need to change the stored value
Current value is closest, no update needed

Bucket P7 is full, checking if we need to change the stored value
Current value is closest, no update needed

Bucket P14 is full, checking if we need to change the stored value
Current value is closest, no update needed

Refresh k-Bucket operation is performed

My DHT:
[ P5, 127.0.0.1:60938, 02a06281]
[ P7, 127.0.0.1:60935, 6f507657]
[ P14, 127.0.0.1:60944, 7e4a630f]
```

Since peer2 and peer 3 were also in the DHT, peer 2 received:

```
Received Hello Message from peer4 7e4a630f along with DHT
[127.0.0.1:60938, 02a06281]
[127.0.0.1:60935, 6f507657]
[127.0.0.1:4984, 7cf2cf17]

Bucket P5 is full, checking if we need to change the stored value
Current value is closest, no update needed

Bucket P7 is full, checking if we need to change the stored value
```

```
Current value is closest, no update needed

Bucket P7 is full, checking if we need to change the stored value
7e4a630f is closer than our current stored value 7cf2cf17, therefore we will update.

Refresh k-Bucket operation is performed

My DHT:
[ P5, 127.0.0.1:60938, 02a06281]
[ P7, 127.0.0.1:60944, 7e4a630f]
```

And peer 3:

```
Received Hello Message from peer4 7e4a630f along with DHT
[127.0.0.1:60938, 02a06281]
[127.0.0.1:60935, 6f507657]
[127.0.0.1:4984, 7cf2cf17]

Bucket P5 is full, checking if we need to change the stored value
Current value is closest, no update needed

Bucket P5 is full, checking if we need to change the stored value
Current value is closest, no update needed

Bucket P5 is full, checking if we need to change the stored value
Current value is closest, no update needed

Refresh k-Bucket operation is performed

My DHT:
[ P5, 127.0.0.1:60935, 6f507657]
```

6 Submission

- Your source code should be fully commented and formatted.
- Your project folder structure should be created such that all the code is under a folder called peer
- Compress these two folders in an archive file (zip file) and name it yourUWOID-SE3314b-assignment2.zip.
- Submit this zip file on OWL by the due date mentioned above.
- Information regarding the Demo and marking scheme will be communicated in a separate file.