Three main components:

|  |
| --- |
| FileShare, Kademlia Services |
| RPC Layer |
| Network Layer |

**Network Layer:**

* All networking code is abstracted

Example of networking class that always responds to hello:

class MessagePrinter extends MessageListener

(needs to implement the following)

void onMessage(IncomingMessage message){

str = message.getDataString(); //two components of the message

URI = message.getSenderURI();

message.sendResponse(str); //quick response to any message received

}

To be able to send a message without getting a response first:

class MessageSender

void sendData(URI destination, String data);

class MessageServerHandler

static MessageSender startUDPServer(int portListen, MessageListener ml);

A **URI** is structured in the following way:

* protocol://hostname:port

**RPC Layer:**

The purpose of the RPC layer is to be able to interact with the network layer and provide a high level request/response RPC mechanism for many services

A message looks like the following:

* {“PBOX”: “1.0”,

“id”: 123, “service”: “kad” (**name of service to route request data to)**, “request”: “” (**String request data)**}

* Various services such as kadmelia or filesharing can register themselves as services the RPC layer with unique service ids

Steps:

1. RPC layer receives data from network layer
2. RPC parses JSON message according to above format
   1. Examines service attribute
3. If service with corresponding service id is registered with RPC layer, RPC layer forwards the request data to the appropriate service
   1. Often the request data will be JSON itself, but it is passed as a string and must be interpreted by the receiving service
   2. When a request is passed to a service, a responder is also provided so the service can directly respond to the request (including wrapping it with the correct id and JSON protocol)
   3. When the RPC layer is used to make a request to a remote destination(URI), a callback must be provided to receive the response or failure.

RPC Handler

* registerServiceListener(String serviceName, ServiceRequestListener listener)
  + a service will implement a ServiceRequestListener, so that all incoming requests addressed to that service will be handled/passed along to the appropriate ServiceRequestListener implementation
* sendRequest(URI destination, String serviceName, String data, RPCResponseListener listener)

class ServiceRequestListener

(needs to implement the following)

void onRequestReceived(RPCEvent e);

An RPCEvent argument has the following:

* String getDataString();
* void respond(String responseData);

RPCResponseListener

* for receiving responses to requests you have sent
* can be implemented inline
* need to implement:
  + void onResponseReceived(RPCEvent e);

**Kademlia Service Layer:**

NetworkInstance

* An instance of a local node that is actively/inactively involved in a particular Kademlia overlay network
* Handles all fundamental Kademlia functionality
  + findNode
  + findValue
  + storeValue
  + ping
* The Kademlia functionality is asynchronous
* These functions are not direct RPC calls but rather complete routing algorithms which “return” the final response or lack thereof via asynchronous callback
* The algorithms utilize RPC layer for actual RPC calls, abstracted from direct network interaction
* Kademlia RPCs have requests/responses encoded as JSON strings

**Example of how to find a node using all layers**:

void findNode(Identifier targetNodeID, ResponseListener<FindNodeResponse> listener)

* ResponseListener receives final result of “find node” operation as a FindNodeResponse object
* Create a FindNodeRequest object
  + request = new FindNodeRequest(getNodeLocalIdentifier(), targetNodeId);
* FindProcess.execute(this, request, ResponseListener<FindNodeResponse>);

void FindProcess.execute(…)

* Create an instance of FindProcess
  + FindProcess<**T**> process = new FindProcesss();
    - **T** is the type of Find Response (i.e FindNodeResonse, FindValueResponse)
      * Gathered from ResponseListener type parameter
  + process.nextIteration();
    - starts the actual find node algorithm

Within FindProcess Algorithm to find node, for example, when making the actual RPC calls:

* networkInstance.sendRequestRPC(Node recipient, FindNodeRequest request, new ResponseListener<FindNodeResponse> **listener**)
  + A new **listener** iscreated inline for each RPC call that is sent out.
  + This is not the same listener that is passed to the execute method, for that is the listener that is called at the end of the algorithm with the final response.

NetworkInstance.sendRequestRPC(…, Request req, ResponseListener<T> listener)

* We use the GSON library to serialize the Java object as a JSON object that is represented as a string
* String requestData = gson.toJson(req)
* getRPC().sendRequest(recipient.getNetworkURI(), “Kad”, requestData, new RPCResponseListener() {void onResponseReceieved(RPCEvent e)}
  + function in Network Instance that returns our RPC handler instance

implementation of inline onResponseReceived is as follows:

* we have the ResponseListener<T> instance from one of the parameters of sendRequestRPC that is surrounding the inline function definition of RPCResponseListener.onResponseReceived
* listener.onResponseReceived(gson.fromJson(e.getDataString(), new TypeToken<T>(){}.getType()))
  + gson requires you to specify the java type you are deserializing too (i.e. FindNodeResponse)
  + getType() returns a java reflection “Type” object that represents type T