```java
package ballmerpeak.turtlenet.server;

import ballmerpeak.turtlenet.shared.Message;
import java.util.Vector;
import java.util.Date;
import java.security.*;
import java.io.*;
import java.net.*;
import java.util.concurrent.Semaphore;

public class NetworkConnection implements Runnable {
    public NetworkConnection (String serverurl) {
        url         = serverurl;
        messages    = new Vector<String>();
        lastRead    = 0;
        messageLock = new Semaphore(1);
        connected   = true;
        tor         = true;

        //parse db/lastread
        File lastReadFile = new File("./db/lastread");
        if (lastReadFile.exists()) {
            try {
                BufferedReader reader = new BufferedReader(
                                            new FileReader(lastReadFile));
                lastRead = Long.parseLong(reader.readLine());
                Logger.write("INFO", "NetCon","Read lastRead from file");
            } catch (Exception e) {
                Logger.write("ERROR", "NetCon", "Could not read lastread from file");
            }
        }
    }

    public void run () {
        Logger.write("INFO", "NetCon","NetworkConnection started");
        while (connected) {
            try {
                Thread.sleep(1000); //update every second
            } catch (Exception e) {
                Logger.write("WARNING", "NetCon", "Sleep interrupted: " + e);
            }
            downloadNewMessages();
        }
    }

    public void close () {
        Logger.write("INFO", "NetCon","close()");
        connected = false;
        try {
            File lastReadFile = new File("./db/lastread");

            if (lastReadFile.exists())
                lastReadFile.delete();

            BufferedWriter writer = new BufferedWriter(
                                        new FileWriter(lastReadFile));
            writer.write(Long.toString(lastRead));
            writer.close();
            Logger.write("INFO", "NetCon","Saved lastRead to disk");
        } catch (Exception e) {
            Logger.write("ERROR", "NetCon", "Unable to save lastRead: " + e);
        }
    }

    //returns true if a message is available
    public Boolean hasMessage () {
        try {
            messageLock.acquire();
            boolean haveMessage = messages.size() >= 1;
            messageLock.release();
            return haveMessage;
        } catch (Exception e) {
            Logger.write("WARNING", "NetCon", "Acquire interrupted");
        }
        return false;
    }

    //get the next message in the queue, and remove it from the queue
    public String getMessage() {
        try {
            messageLock.acquire();
            String m = messages.get(0);
            messages.removeElementAt(0);
            messageLock.release();
            return m;
        } catch (Exception e) {
            Logger.write("WARNING", "NetCon", "Acquire interrupted");
        }
        return new Message("NULL", "", 0, "").toString();
    }

    public long getTime () {
        Vector<String> time = serverCmd("t");

        if (time.size() == 2)
            return Long.parseLong(time.get(0));
        else
            Logger.write("ERROR", "NetCon", "Couldn't retreive time from server");

        return 0;
    }

    public boolean postMessage (Message msg, PublicKey recipient) {
            String ciphertext = Crypto.encrypt(msg, recipient, this);
            if (!serverCmd("s " + ciphertext).get(0).equals("s")) {
                Logger.write("RED", "NetCon", "server reported failure uploading message");
                return false;
            } else {
                Logger.write("INFO", "NetCon", "uploaded message: \"" + msg + "\"");
                return true;
            }
    }

    //The only time unencrypted data is sent
    public Boolean claimName (String name) {
        try {
            Message claim = new Message("CLAIM", name,
                     getTime()+Crypto.rand(0,50), "");
            claim.signature = Crypto.sign(claim);
            String cmd = "c " + Crypto.Base64Encode(claim.toString().getBytes("UTF-8"));
            if (serverCmd(cmd).get(0).equals("s")) {
                Logger.write("INFO", "NetCon","Claimed name: " + name);
                Logger.write("INFO", "NetCon","\tname: " + claim.CLAIMgetName());
                Logger.write("INFO", "NetCon","\ttime: " + Long.toString(claim.getTimestamp()));
                Logger.write("INFO", "NetCon","\t sig: " + claim.getSig());
                return true;
            }
        } catch (Exception e) {
            Logger.write("ERROR", "NetCon", "Could not register name: " + e);
        }

        Logger.write("INFO", "NetCon","Could not register name: " + name);
        return false;
    }

    public void downloadNewMessages () {
        Vector<String> msgs = serverCmd("get " + lastRead);
        lastRead = getTime();

        for (int i = 0; i < msgs.size(); i++) {
            if (!(msgs.get(i) == null) && !msgs.get(i).equals("s") && !msgs.get(i).equals("e")) {
                try {
                    messageLock.acquire();
                    messages.add(msgs.get(i));
                    messageLock.release();
                } catch (Exception e) {
                    Logger.write("WARNING", "NetCon", "Acquire interrupted.");
                }
            }
        }
    }

    //send text to the server, recieve its response
    private Vector<String> serverCmd(String cmd) {
        Socket s;
        BufferedReader in;
        PrintWriter out;
        //if (!cmd.equals("t") && !cmd.substring(0,4).equals("get "))
        //    Logger.write("VERBOSE", "NetCon", "Sending command to server \""  + cmd + "\"");

        //connect
        try {
            if (tor) {
                s = new Socket(new Proxy(Proxy.Type.SOCKS,
                                         new InetSocketAddress("localhost", 9050))); //connect to Tor SOCKS proxy
                s.connect(new InetSocketAddress(url, port));                         //connect to server through Tor
            } else {
                s = new Socket(url, port);
            }

            in  = new BufferedReader(new InputStreamReader(s.getInputStream()));
            out = new PrintWriter(s.getOutputStream(), true);
        } catch (Exception e) {
            Logger.write("ERROR", "NetCon", "Could not connect to network: " + e);
            return null;
        }

        //send command
        out.println(cmd);
        out.flush();

        //recieve output of server
        Vector<String> output = new Vector<String>();
        try {
            String line = null;
            do {
                line = in.readLine();
                if (line != null)
                    output.add(line);
            } while (line != null);
        } catch (Exception e) {
            Logger.write("ERROR", "NetCon", "Could not read from rserver: " + e.getMessage());
        }

        //disconnect
        try {
            out.close();
        } catch (Exception e) {
            Logger.write("ERROR", "NetCon", "Could not disconnect from rserver: " + e.getMessage());
        }

        try {
            in.close();
        } catch (Exception e) {
            Logger.write("ERROR", "NetCon", "Could not disconnect from rserver: " + e.getMessage());
        }

        try {
            s.close();
        } catch (Exception e) {
            Logger.write("ERROR", "NetCon", "Could not close socket: " + e.getMessage());
        }

        return output;
    }

    private String url;
    private final int port = 31415;
    private Vector<String> messages;

    private long lastRead;
    private boolean connected;
    private boolean tor;
    private Semaphore messageLock;
}
```