CS5450 Lab3 P2P Report

|  |  |  |
| --- | --- | --- |
| Bo Gao  Bg447@cornell.edu | Zexi Liu  Zl558@cornell.edu | Jacqueline Pang  Jp2338@cornell.edu |

Abstract

# No central server is needed, every peer receives the message and randomly forwards it to a neighbor, and there are anti-entropy mechanism to ensure every message can propagate to the whole system.

# 1. Design Principle

The file netsocket.cc/ netsocket.hh are designed to handle netsocket init issues such as randomly select neighbors and bind.

Other functions are considered implemented under ChatDialog, so implemented in main.cc including UI/timer initialization, sending receiving rumor messages and status messages. Function names are defined properly based on their usage.

# 2. Highlighted Implementation Details

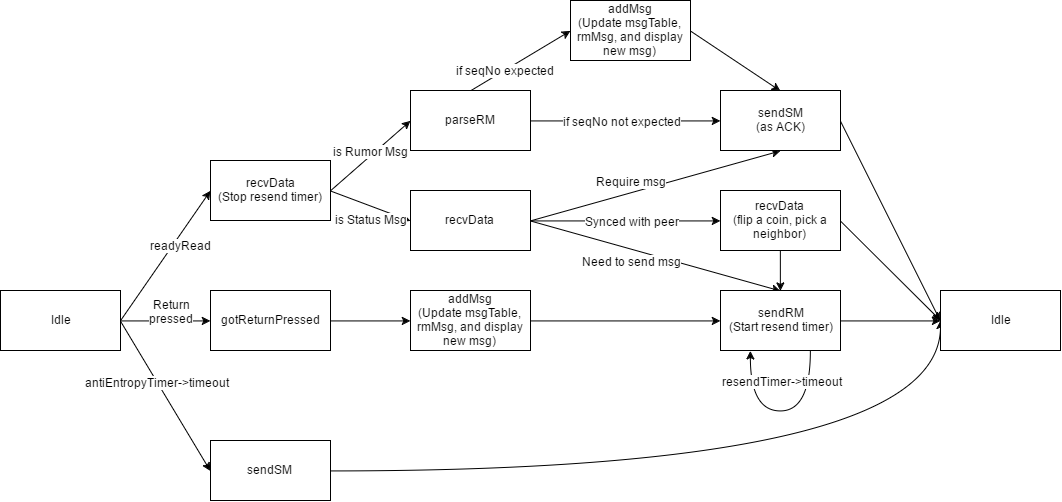


FIG. 1. Design Logic

2.1 Initialization

Modify UI to better monitor the sending hostname, port number and message ID. Debug msg are resonated at background.

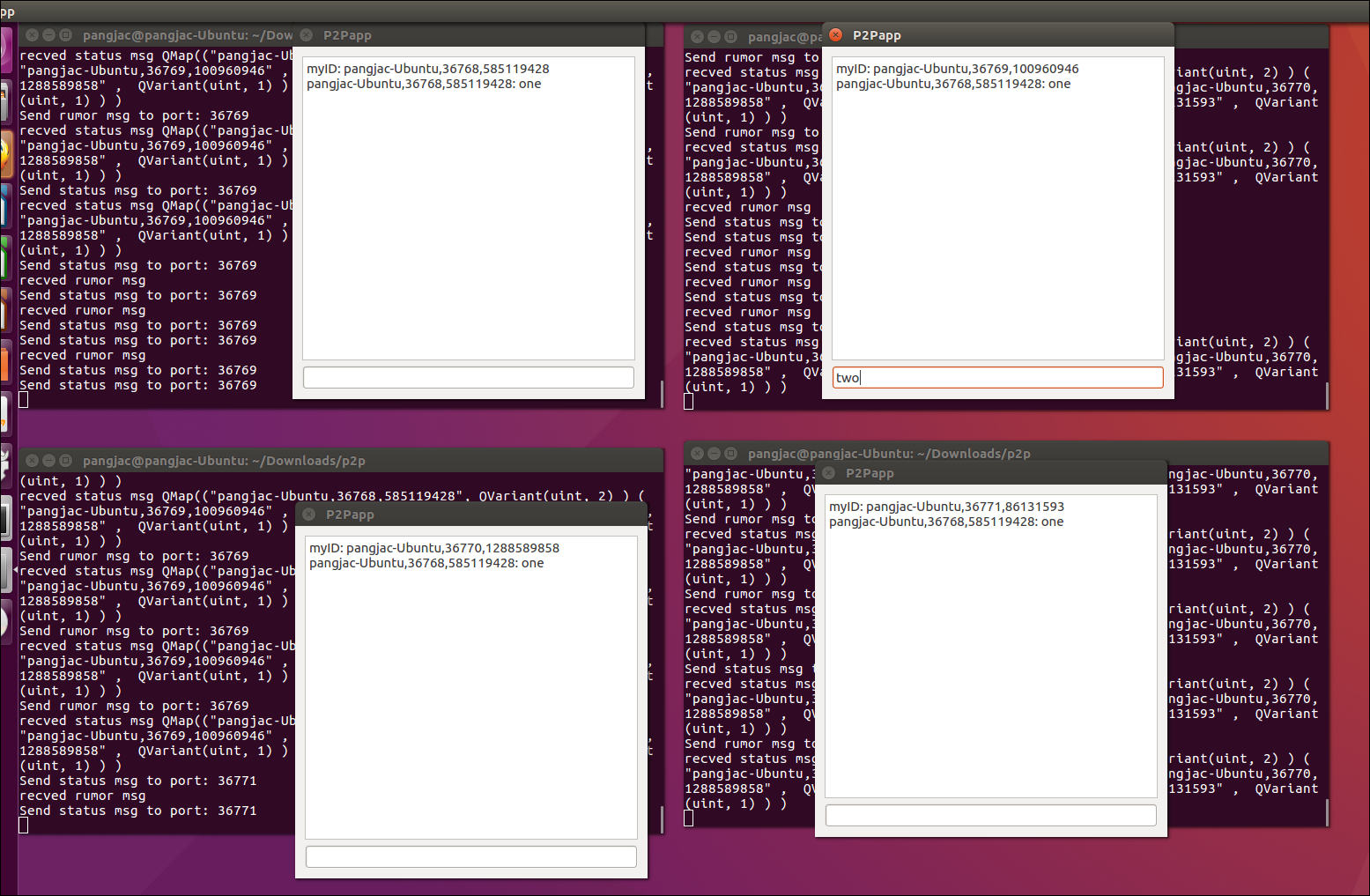
2.2 . Implementation

I. sendRM method use parameters to differentiate whether to send messages or resend messages. If there is no given valid parameters (i.e. port <= 0), then this is to resend message, and the way to call it is based on the resendTimer .

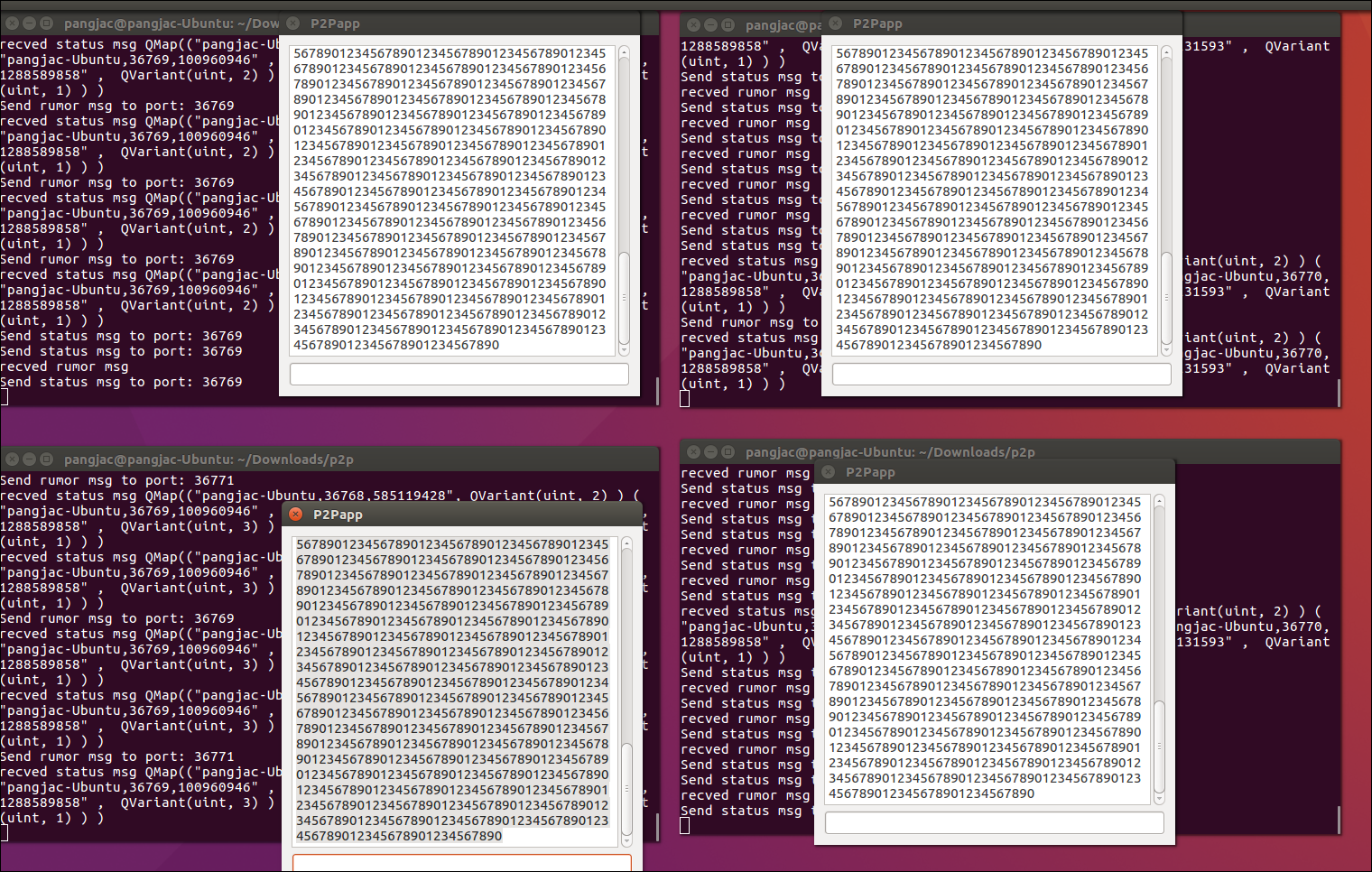
II. We use QMap variant msgTable to record the message ID and message info. Once got a message, check its associated messageID and if it is not in msgTable, resend the message.

# 3. Test Steps and Sscreenshot

* 1. Internal Test
     1. Simple string :



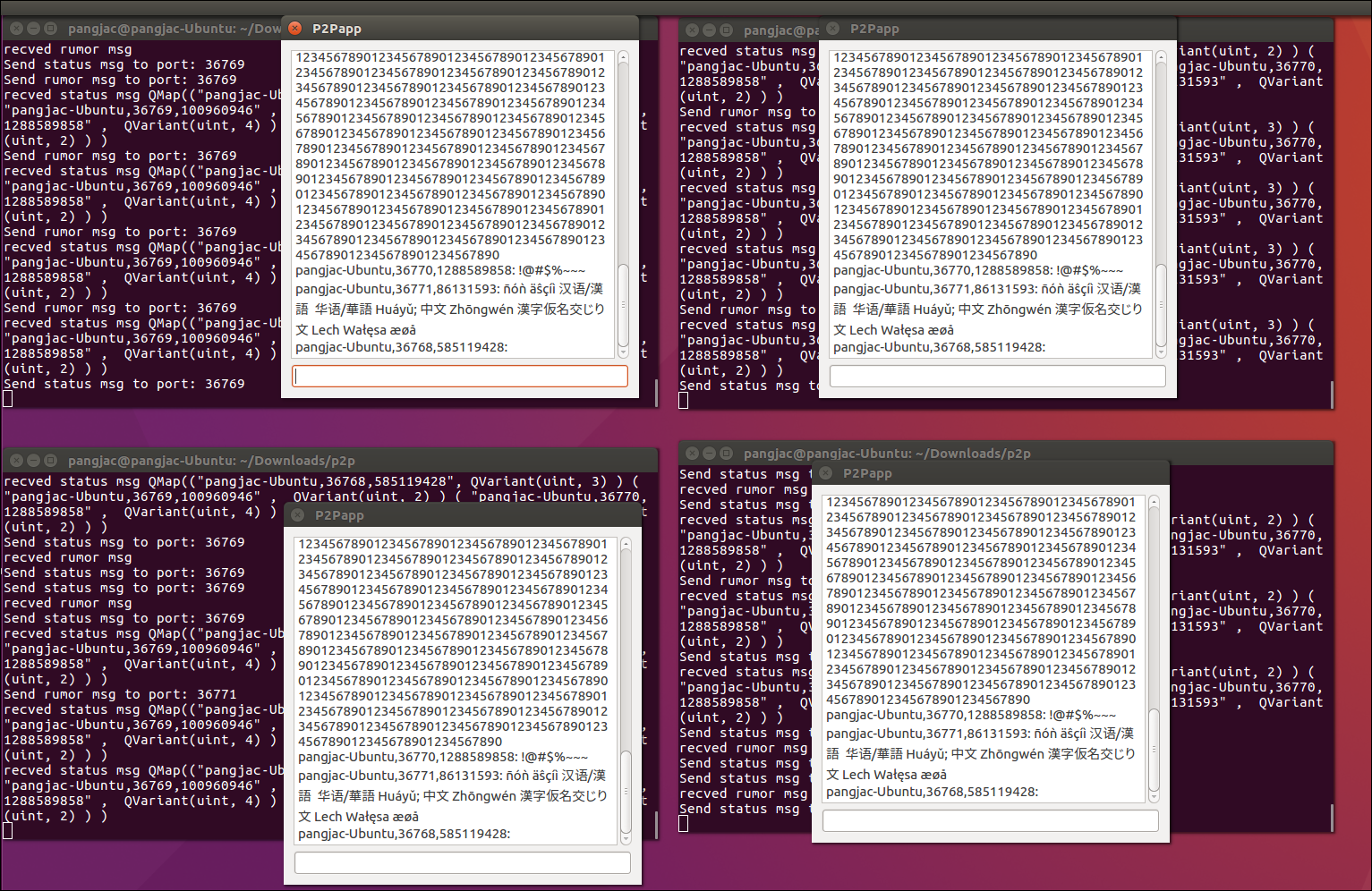
* + 1. Send long number:



* + 1. Send special chars:

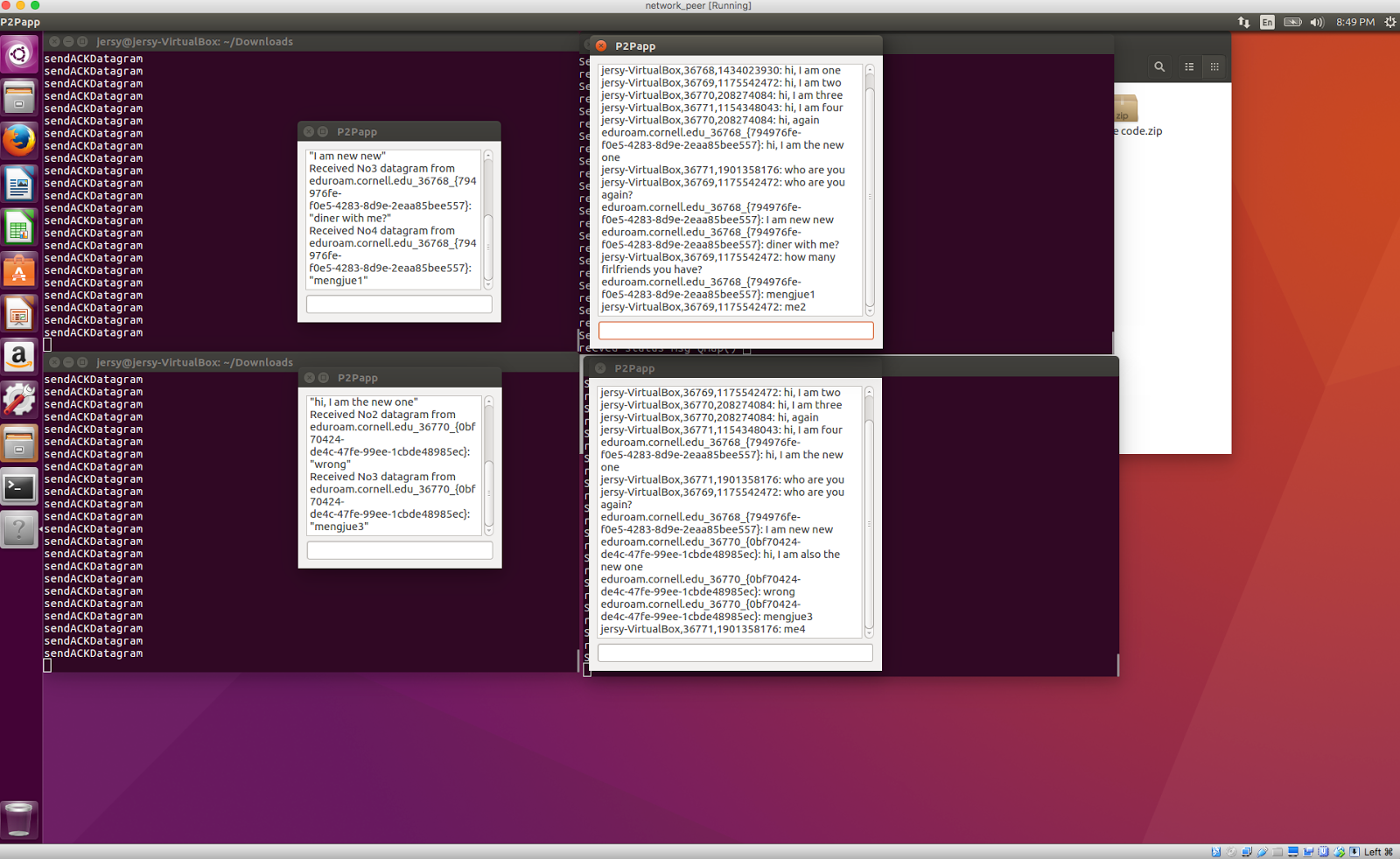


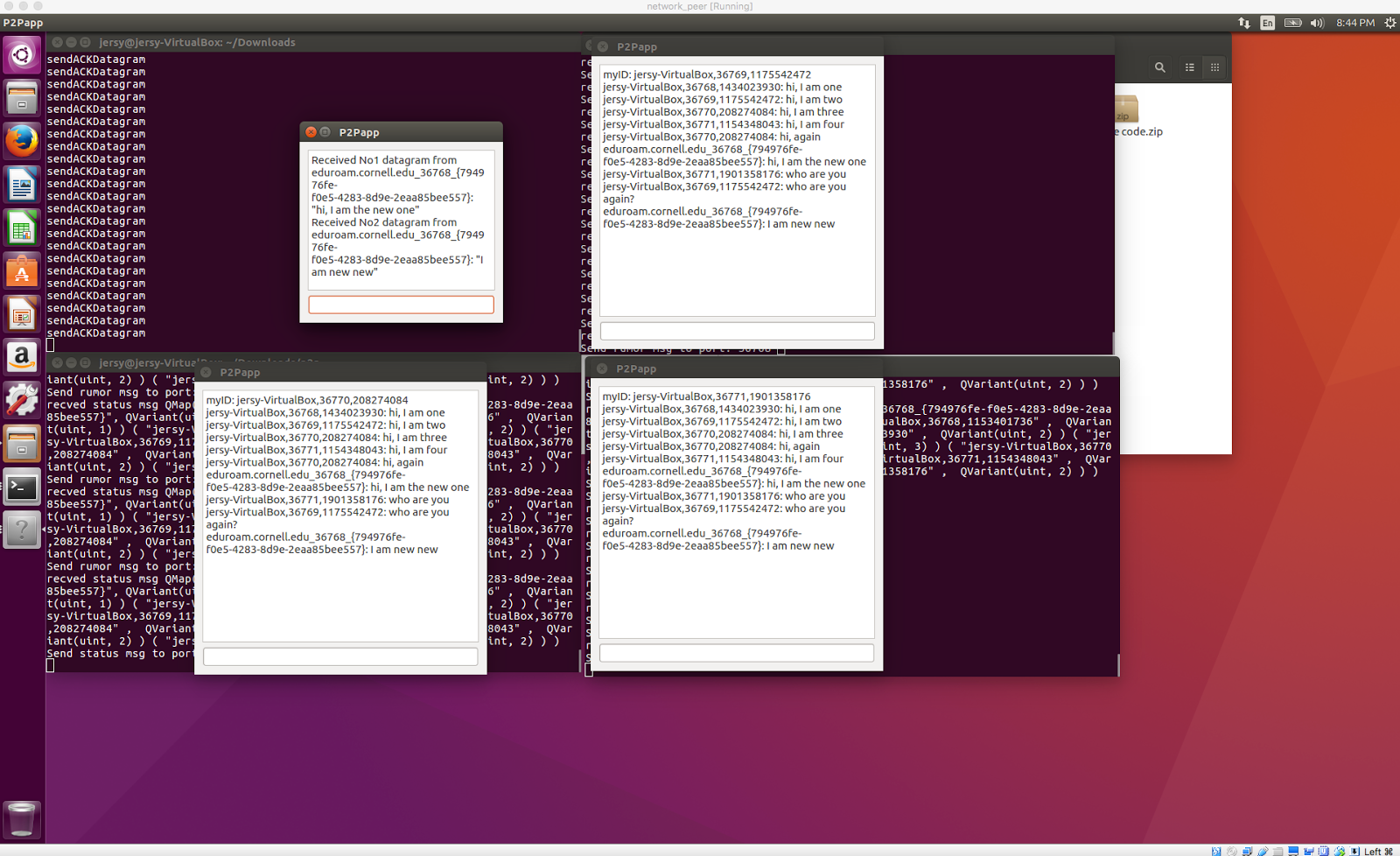
* + 1. Empty Lines



* 1. Cross Team Test

3.2. 1 Message can only be send uni-lateral : app 1& 3 are from Mengjue’s team; 2&4 are from our team. We can see that message “mengjue1” can send from app1 to 2, but break to app 3, while app 3 could send message “mengjue3” can send to app4, but cannot send to 1 and 2.



3.2.2 While if app3 is also from our team’s, then message could be send out.

# 4. Vulnerability Test

4.1 Handle “DOS Attack”

If any participant do not behave properly, it may intentionally send repeated status messages to one host such that the target host will have to resend message and thus its bandwidth would be occupied.

4.2 Handle “Man-in-the-middle Attack” :

Right now this implementation does not recognize the participants’ identification, so everyone in the link can get messages.

Even though we have a mechanism to recognize participants, we could meet Man-in-the-middle Attack problem, when the attacker impersonate one of endpoints. We can prevent this attack by adding SSL authentication protection – forward messages by using a mutually trusted certificate of authority.

4.3 Zero-based status message conflicts

Right now, the implementation message ID starts from 1 and increased by 1 for every new message, i.e we are doing One-based status protocol. It is possible that a zero-based status message could not be recognized properly, and thus leads to one or more endpoints constantly re-send the same messages , result in system traffic.

4.4 Balancing between consistency and performance

Right now, it is obvious having messages delay when spreading to endpoints. Anti-entropy mechanism can help to ensure that every message propagate to the whole system, but it is also the reason of inconsistent. To make sure all nodes receive all message, we set an entropy timer for 10 seconds. We could increase performance, by sacrificing message traffic in the system (since timer is shorter, so higher possibility to resend the message to the whole system).

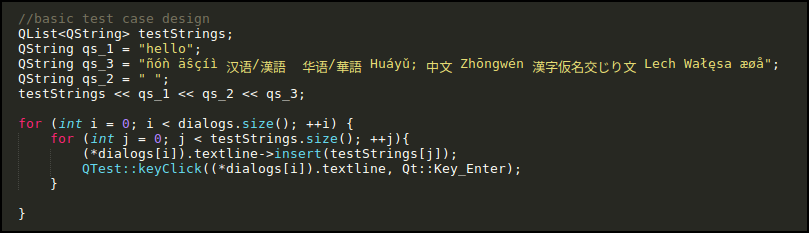
# 5. Automation Test

5.1 Design and How to use

Add file test.cc and test.hh to the original submission.

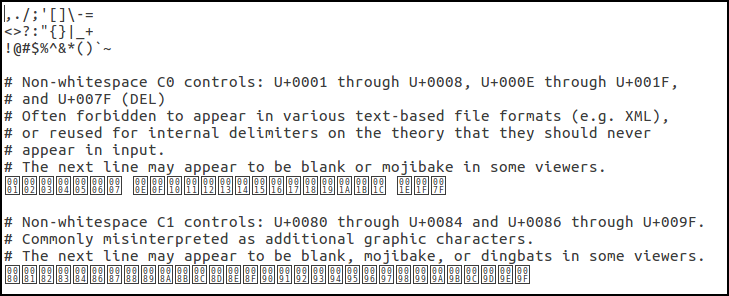
Total 4 ChatDialogs would be created. This is designed at method testGui(), where we can see that we define 4 dialogs.

A basic test, including a simple string, an empty string, and a string conclude non ascii, char is designed as this:



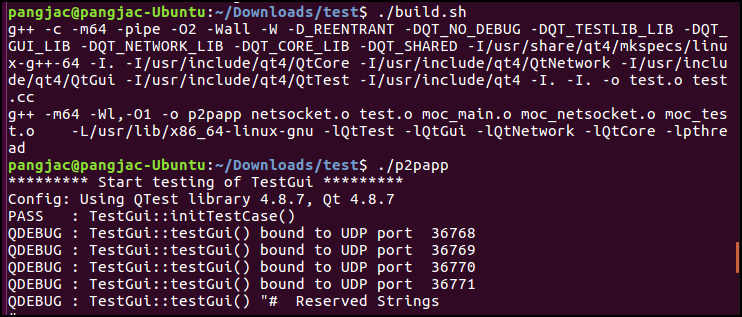
We then extend this basic test to a much more complex one: we combine all complex test cased into one single file, read in and test. The file read into the test is “hell.txt” (which comes from <https://raw.githubusercontent.com/minimaxir/big-list-of-naughty-strings/master/blns.txt> )

A screenshot is like this:



5.2 Expected result

Once you build and run (you may need to run make clean firstly), the read test file content would be also resonated at the terminal.



The expected pass result should be like this:

