Project 1
CS505
2/1/2014

Reliable Channel: Java Implementation

Derek Schatzlein and Josh Reese

Structure –

For this project, we have implemented the interfaces provided in the handout in the following files: RChannel.java, RChannelReceiver.java, and RMessage.java. We have also added RMessageComparator.java (which allows RMessages to be compared for the purposes of putting them into collections), as well as ReceiveThread.java and SendThread.java, which implement threads that will be spawned at boot time to handle the receiving and sending of messages. There is also Node.java, which contains a main method meant for testing. We have prepared javadocs for everything: please build them if further clarification is needed.

To be reliable, messages sent in our channel are subject to several conditions. Firstly, every message has a sequence number and an ACK character prepended to it as a string. RMessage contains getters and setters for these fields. As well, every message that is sent must be ACKed by the receiver. If no ACK is received within three seconds of sending a message, that message will time out, and the sender will resend it accordingly.

In order to have sending be non-blocking, we have implemented *rsend()* such that all it does is put an event into an event queue (called messageQueue). The sender thread (defined by SendThread) will continuously be checking the queue for new events, and will send messages out as they are put into the queue, or have timed out. Receiving works in a similar way. To make *rlisten()* non-blocking we simply spawn a receiver thread (defined by ReceiveThread) which will be waiting to receive messages. Upon receiving a message, the receiver will form it into an RMessage and pass it to the RChannelReceiver. The RChannelReceiver will then parse the message. If the message is an ACK, then it will put an event onto a queue called ackList, which is to signal the sender that a message has been ACKed so that it can be removed from timeout management. If the message was not an ACK, then it will put an event into the toAck queue which contains messages that have not received ACKs yet. This will also be used by the sending thread to send ACK's. Received messages will be written to a file called output.out.

Usage – (see Readme for more detailed information)

For compilation, we have used ant to make everything simple. Simply type:

```
ant jar
```

and everything will build. For more detailed information, see the README.md file, located in the same directory as this report.

Node.java contains our testing suite. It reads in messages (separated by new-lines) from standard in (we piped a file to it from the command line). It then initializes everything, spawns the sending and receiving threads, and calls rsend() for each message to put a “send” event on the messageQueue. Node takes two command line arguments: the first is the IP Address of where you would like to be sending messages, and the second is a machine ID. This machine ID exists solely to differentiate two nodes from each other if they are running on the same machine. The machine ID can only be 0 and 1. An example run would look like this:

```
cat data.txt | java -jar dist/rchannel.jar localhost 0
```