Ex 3 Design Document

Overview:

Since we are using spread, which takes care of the low level details of agreed order, we are only concerned with the coase grain flow control to determine what/when/how processes send, receive, and exit. In this protocol a process is only allowed to send a certain number of packets at a time. To enforce flow control it may not send more packets until it has read and delivered all the packets that it last sent. Senders who have no packets to send or are completed will call receive until all processes are finished.

This way, processes will not block for long periods of time because when any process calls sp_receive, there is at least one message that has already been sent across the network. Additionally, the window that each process is allowed to send can be tuned easily to allow the highest possible throughput without overflowing spread's internal buffers.

Data Structures:
Array FINISHED - to keep track of which processes are finished sending, once all processes are finished, we can safely exit

Message structure as highlighted in exercise, 1312 bytes from 3 ints and 1300 bytes of data

```
struct Message
{
      int process_index
      int message_index;
      int magic_number;
      char data[1300];
};
```

Protocol:
Let n be the number of machines
m be number of messages a process needs to send

Initialize:
DONE_SENDING = false
LAST_SENT = 0
PACKET_BURST_SIZE = b // Tunable window size parameter
N_finished = 0

```
Call receive until we receive a membership message where the number of members is n
If m = 0, DONE = True
Repeat while not DONE_SENDING:
        Call send_packet_burst
        Repeat indefinitely: // receive messages until we've read all messages we have sent
                Receive and deliver message
                If the message sender is us and the index is LAST_SENT:
                        Break

Repeat:
        Receive and deliver message
        If message.message_index = -1
                N_finished++ // count another process that is done sending
                If n_finished = m:
                        End protocol


Subroutine Send_packet_burst():
        For i = LAST_SENT + 1 to LAST_SENT + b
                Send packet i
                If i = n
                        Send packet -1
                        LAST_SENT = -1
                        DONE_SENDING = true
                        return
        LAST_SENT = LAST_SENT + b
        return
```

Discussion:

Our protocol was able to complete in an average of 30s, which is equivalent to a transmission rate of about 330 Mbit/s. The default window size (PACKET_BURST_SIZE) is 75, although we were able to successfully run the protocol with higher window sizes.

A higher window size translates to better performance, so it is beneficial to run with as high a window size as possible without the program crashing. A few times, our program crashed with a window size of 100 or more. We are unsure if this is due to our code or network traffic, so we have included a makefile option to customize the window size easily.

Running the program:

To run the program, first make sure the project directory is clean by running ‘make clean’. To build the project with a default window size of 75, run ‘make’. To run with a window size of

<WINDOW>, run the command ‘make window=<WINDOW>’. The program can be invoked according to the assignment’s specifications.