HW#4 (Optional)

In this homework we reuse the same input files in HW#2.

This is a sequel of HW2b. We inherit the same assumption (anonymous algorithm plus randomized initial values; M1 plus M2) and the same decision making rule. However, we consider a different execution model where message propagation takes time and there is no synchronization scheme (e.g., synchronizer) to make program executions synchronized (running in a round basis). Due to the lack of shared-variable infrastructure, p_i's decision variable (whether to join the MIS or not) may not be readily available to other processes. So p_i should send the updated decision to every neighbor (sending one message to each neighbor) every time p_i changes its decision. Messages delays here are independent and identically distributed random variables. After receiving an update from p_i, the receiver process updates its cached version of p_i's decision variable and may change its decision as well as a response. If the receiver does change its decision, it should also send its new decision to every neighbor of it.

Each process p_i maintains an event queue Q_i to keep all local events to be processed. Every event is associated with a timestamp (a real number) and all events in an event queue are always kept in an increasing order of timestamps. Initially, every process sets its decision variable to a value other than 0 or 1, sets all locally cached versions of other process's decision variables to an arbitrary value, and inserts an event with a timestamp randomly determined with the range [0, 10] into its local event queue.

Your simulation program (i.e., the only god or daemon here) uses a global time T to determine the next event to execute. The global time is only used by the simulation program; processes have no knowledge of it. The next event to execute will be the one with the smallest timestamp (the one that is closest to the current global time.) Since all events in an event queue are sorted by their timestamps, the next event to execute must be at the head of some event queue. So the simulation program just examines every head event to find it. Let ts_i be the timestamp of the head event in event queue Q_i. The next event to execute will be the one with timestamp $\min_j\{ts_j\}$.The simulation program then sets T to be the timestamp of that event and executes it.

Executing an event is to give the process a chance to revise its decision. If the process decides not to change its decision, then nothing happens. The simulation program just proceeds to the next event. If the process does change its decision, it should send its decision to each of all its neighboring processes. The simulation program simulates a message sending with an arbitrary delay by inserting a receiving event to the receiver's event queue. Suppose that at global time $T = t$ the sender p_i sends a message to destination p_j, the simulation program adds a receiving event with timestamp $t + d_{ij}$ at p_j's event queue Q_j, where d_{ij} is an exponentially distributed random variable with mean $1/\lambda$. Note that all d_{ij}'s are independent and identically distributed random variables, which means that these variables follow the same exponential distribution (with same parameter), but their values are independently generated. After inserting all the receiving events to all the destinations, the simulation program proceeds to the

next event. Note that the initial decision of every process is neither 0 nor 1, so the very first event of every process will definitely cause a message sending from the process to each of its neighbors.

The simulation ends when there is no more event to execute in any event queue. Call the new program HW4.

(a) Run HW4 1,000 times on the same input data file and answer the following questions. ① Is it possible that HW4 does not stop for some input? Why? ② If HW4 does stop for some input file, please list all possible results (one line for each set of duplicated results) with respective percentages. Are all these results correct (independent sets)?

(b) Consider two performance matrices: ① averaged total weight (in the set) and ② averaged convergence time (the number of decision changes to the end of the simulations). Change the value of λ in (b) from 0.1 to 0.9 (with step 0.1) and run HW4 1,000 times on the same input data file for each possible setting of λ. Collect the statistics. (a) Use an x-y graph to show how ① changes with the setting of λ. (b) Use an x-y graph to show how ② changes with the setting of λ.

Note: Some programming environment already provides some means to generate an exponentially distributed random variable. In case you don't have that function, please refer to the function written in C here: http://www.cs.nctu.edu.tw/~lhyen/acn/normal.c