**Byzantine fault tolerance**

* A Byzantine fault is any fault presenting different values to different processes
* A Byzantine failure is when a system requires consensus but due to a Byzantine fault cannot be achieved

**Byzantine Generals’ Problem**

Background Story: A group of generals each command a portion of the Byzantine army encircling a city. These generals wish to formulate a plan for attacking the city. In its simplest form, the generals must only decide whether to attack or retreat. Some generals may prefer to attack, while others prefer to retreat. The important thing is that every general agrees on a common decision, for a halfhearted attack by a few generals would become a rout and be worse than a coordinated attack or a coordinated retreat.

The problem is complicated by the presence of traitorous generals who may not only cast a vote for a suboptimal strategy, they may do so selectively. For instance, if nine generals are voting, four of whom support attacking while four others are in favor of retreat, the ninth general may send a vote of retreat to those generals in favor of retreat, and a vote of attack to the rest. Those who received a retreat vote from the ninth general will retreat, while the rest will attack (which may not go well for the attackers). The problem is complicated further by the generals being physically separated and having to send their votes via messengers who may fail to deliver votes or may forge false votes.

**Weighted Byzantine Agreement Problem (WBA)**

* *Github Repo with weighted byzantine agreement algorithm (King and Queen):* [*https://github.com/bethrichardson/dynamic-weighted-byzantine-agreement*](https://github.com/bethrichardson/dynamic-weighted-byzantine-agreement)

***Introduction***

* Without weights consensus can only be reached if for N processes, N is greater than 3 times the number of failures
* In WBA each process P is given a weight w. 0 ≤ w ≤ 1
* The sum of all weights is equal to 1
* Consensus can be reached when the processes that fail is more than N/3 so long as the total weight of the failed processes is less than 1/3
* Number of rounds to reach consensus is less than or equal to the number of process that failed f plus 1 (f + 1 rounds)
* *ρ –* sum of weights of all failed processes
* Anchor of system is the least number of trusted processes whose weight are greater than *ρ*
* weighted-Queen algorithm works when *ρ* < 1/4
* weighted-King works when *ρ* < 1/3
* weight update method reduces suspected faulty processes whose weight is at least 1/4 to 0.

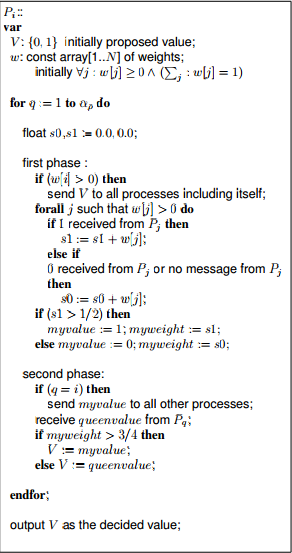
***System Model***

* System is assumed synchronous – there is upper bound on message delay and duration of actions
* Processes may fail but communication is reliable and satisfies FIFO
* *ρ –* sum of weights of all failed processes
* Processes that never fail are called correct processes
* There is a non-negative weight assigned to every process
* All processes have knowledge of the weights of other processes
* The sum of the weights equals 1
* All processes propose a binary value with goal of deciding on one common value
* Agreement – two correct processes cannot decide on different values (They can propose different values though)
* Validity – The value decided must be proposed by a correct process
* Termination – all correct processes decide in finite number of steps

***Weighted-Queen Algorithm***

* Takes α rounds each with **two** phases
* Requires *ρ* < 1/4
* Each process has a preference for each round which is initially its input value
* Processor Pi is the queen for round I (Queen rotates around the ring of processes)
* There will be at least one round where the queen is correct
* **Phase 1** 
  + Every process exchanges its value with all other processes
  + Based on values received and weights of the other processes process estimates and puts this value in myvalue
* **Phase 2**
  + Process receives value from the queen
  + If no value (queen failed) process assumes 0
  + Decide whether to use own value or queenvalue
    - If sum of the weights of processes which proposed myvalue given by variable myweight is greater than 3/4 then myvalue is chosen for V else queenvalue is used
    - This ensures that if all correct processes prefer value v then they continue to do so

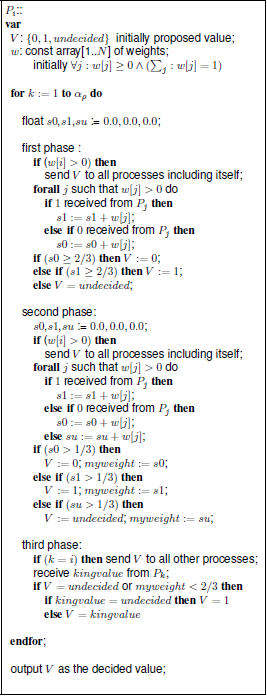
*Algorithm:*



***Weighted-King Algorithm***

* Takes α rounds each with **three** phases
* Requires *ρ* < 1/4
* Each process has a preference V which can be 0,1, or undecided
* Rotating King same as Queen algorithm
* **Phase 1**
  + If process has positive weight then sends V to all processes including itself
  + Process adds weights from all processes that propose 0 to s0 and 1 to s1
  + If s0 or s1 is greater than or equal to 2/3 then process sets its preference V to that value
  + Else process sets preference V to undecided
* **Phase 2**
  + Process sends new V to every process if weight is positive and resets s0, s1, and su (processes may have undecided values)
  + Process adds weights from all processes that have values 1,0, undecided to s0, s1, su
  + If s0,s1,su > 1/3 set preference V to that value if more than one weight is > 1/3 then preference given to 0,1, then undecided
  + myweight is set to cumulative weight of V
* **Phase 3**
  + King for round sends preference to every process
  + Process value V becomes kingvalue if myweight myweight < 2/3 or undecided

*Algorithm:*



***Updating Weights***

I don’t know if we want to do this basically if you’re going to keep using WBA you can start to identify processes that you suspect are faulty and start to assign them lower weights. Based on the project description I don’t think it’s necessary to implement this