Exercise 11

1 Fail-noisy consensus

The implementation of uniform consensus in the fail-noisy model presented in class (Section 5.3 [CGR11]) assumes that $N > 2f$, i.e., there is a majority of correct processes. Recall that it used an eventual leader detector at its core. The fail-noisy model considers processes subject to crashes and assumes an eventually perfect failure detector ($\Diamond P$, Module 2.8) or an eventual leader detector ($\Omega$, Module 2.9).

The “Flooding Uniform Consensus” (Algorithm 5.3, Section 5.2.2), on the other hand, is in the fail-stop model, uses a perfect failure detector ($P$, Module 2.6), and tolerates any number of crashed processes.

Explain why any algorithm in the fail-noisy model that implements consensus also requires a majority of correct processes.

2 Replicated State Machine

Consider a fault-tolerant service, implemented by a replicated state machine. The replicated state machine runs on all $N$ processes in $\Pi$ and might look like this (Module 6.12 of [CGR11, p. 330]):

Module 1: Replicated state machine

Module:

   Name: ReplicatedStateMachine, instance rsm.

Events:

   Request: ( rsm, Execute | command ): Requests that the state machine executes the command given in command.

   Indication: ( rsm, Output | response ): Indicates that the state machine has executed a command with output response.

Properties:

   RSM1: Agreement: All correct processes obtain the same sequence of outputs.

   RSM2: Termination: If a correct process executes a command, then the command eventually produces an output.

There are multiple clients that invoke operations on the service and expect to receive a response. Which extensions are needed to the replicated state-machine abstraction so that
clients can access the service and receive correct responses from the service? For instance, how do clients associate responses to commands?

What changes if the replicated state-machine abstraction is implemented by Byzantine processes?