# CSC 2221 HW3

Eric Di Tomasso

Model: Synchronous message passing, $n$ processes $p_0, ..., p_{n-1}$. Let $P$ be the set of all processes. Each process $p_i$ receives an input $x_i$.

## 1 Impossibility Result

**Lemma 1.** *It is impossible to solve Byzantine Agreement deterministically in a synchronous message passing network when up to $f$ processes can be Byzantine, if the network is not $(2f+1)$-connected.*[1]

*Proof.* Suppose there is such an algorithm. Since the network is not $(2f+1)$-connected, there 2 sets of processes $S_0$ and $S_1$ such that there are at most $2f$ process disjoint paths between them in the network. Thus, there is a set or processes $C$ of size at most $2f$ such that all messages sent between $S_0$ and $S_1$ pass through at least 1 process in $C$. Let $C_0$ and $C_1$ be two disjoint sets such that $C_0 \cup C_1 = C$, $|C_0| \leq f$, and $|C_1| \leq f$. Let $\beta$ be an execution of the algorithm on the ring pictured in Figure 1. Note that in the ring, each node corresponds to a set of processes, and an edge between 2 nodes corresponds to the set of all edges joining processes between the two sets. The number within each node corresponds to the input value for each process in the set of processes that the node represents. Consider the following 3 scenarios.

**Scenario 1:** Let $\alpha_1$ be an execution of the algorithm in a network in which all processes start with input 0, and all processes in $C_1$ are faulty. Assume processes in the set $C_1$ are sending to $S_0$ the messages sent in $\beta$ by $C_1$, and are sending to $C_0$ and $S_1$ the messages sent in $\beta$ by $C_1'$. This scenario is illustrated in Figure 2.

**Scenario 2:** Let $\alpha_2$ be an execution of the algorithm in a network in which all processes in $S_1$ start with input 0, all other processes start with input 1, and all processes in $C_0$ are faulty. Assume $C_0$ is sending to processes in $S_0$ the messages sent in $\beta$ by $C_0'$, and is sending to all processes in $C_1$ and $S_1$ the same messages as those sent in $\beta$ by $C_0$. This scenario is illustrated in Figure 3.

**Scenario 3:** Let $\alpha_3$ be an execution of the algorithm in a network in which all processes start with input 1, and all processes in the set $C_1$ are faulty. Assume processes in the set $C_1$ are sending to $S_0$ the messages sent in $\beta$ by $C_1'$, and are sending to all neighbours in $C_0$ and $S_1$ the messages sent in $\beta$ by $C_1$. This scenario is illustrated in Figure 4.

I now argue that $\alpha_1 \overset{S_1}{\sim} \alpha_2$. Notice that in $\alpha_1$, processes in $S_0$ have the same initial configuration and receive the same messages from $C_1$ as they do in $\beta$. The same claim holds for the sets $C_0$ and $S_1$. Thus, for all correct processes, the execution $\alpha_1$ is indistinguishable from the execution $\beta$. In $\alpha_2$, processes in $S_0$ have the same initial configuration and receive the same messages from $C_0$ as processes in $S_0'$ do in $\beta$, and processes in $C_1$ have the same initial configuration and receive the same messages from $C_0$ as $C_1'$ in $\beta$. Thus in $\alpha_2$, $S_1$ has the same initial configuration it does in $\beta$, and receives the same messages it does in $\beta$. This is due to the fact that in both executions, the actions of the set of Byzantine processes are defined with respect to $\beta$. Hence, $\alpha_1 \overset{S_1}{\sim} \beta \overset{S_1}{\sim} \alpha_2$, so we conclude that $\alpha_1 \overset{S_1}{\sim} \alpha_2$. A similar argument can be used to show $\alpha_2 \overset{S_0}{\sim} \alpha_3$. By validity, all non-faulty processes in Scenario 1 must decide 0. Because $\alpha_1 \overset{S_1}{\sim} \alpha_2$, all processes in $S_1$ must also decide 0 in $\alpha_2$, and by agreement all processes in $S_0$ must decide 0 in $\alpha_2$. Applying indistinguishability again, it can be observed that all processes in $S_0$ must therefore decide 0 in $\alpha_3$. This violates validity, and is a contradiction.

□

[1] The high level approach for this proof can be found in course notes at the following URL: http://www.cs.yale.edu/homes/aspnes/pinewiki/ByzantineAgreement.html

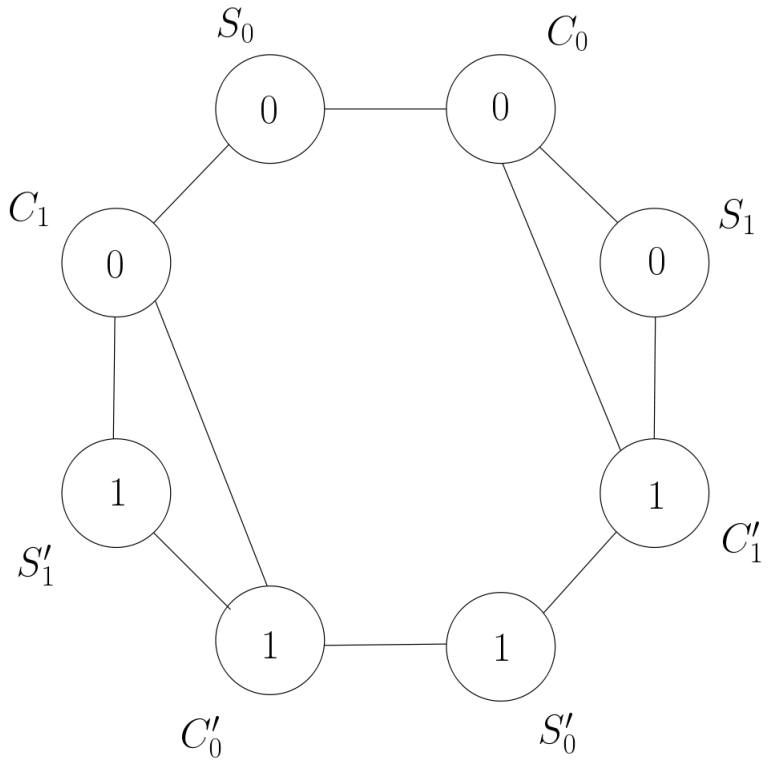


Figure 1: The ring of processes.

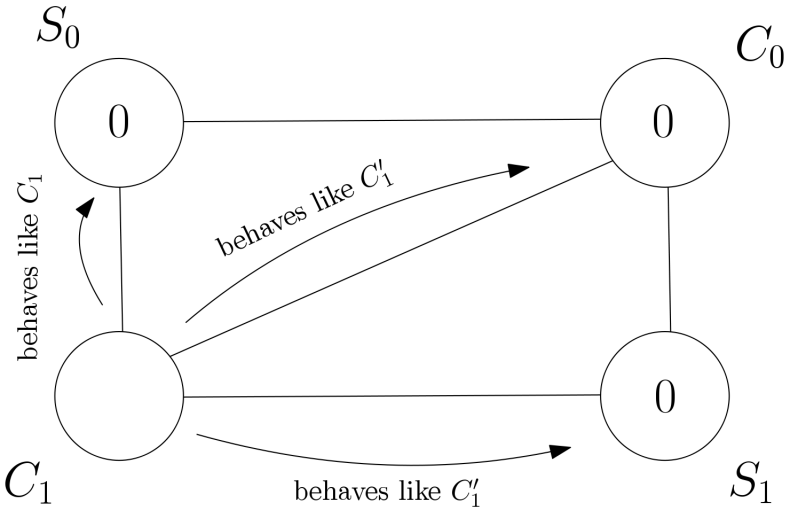


Figure 2: Scenario 1.

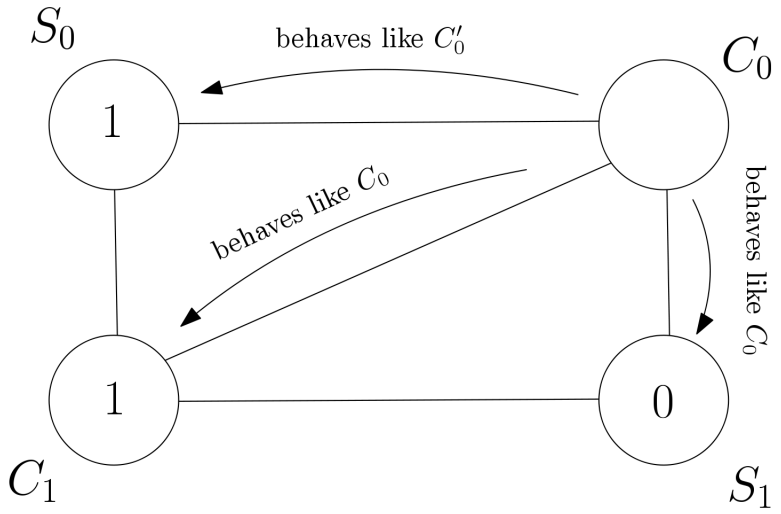


Figure 3: Scenario 2.

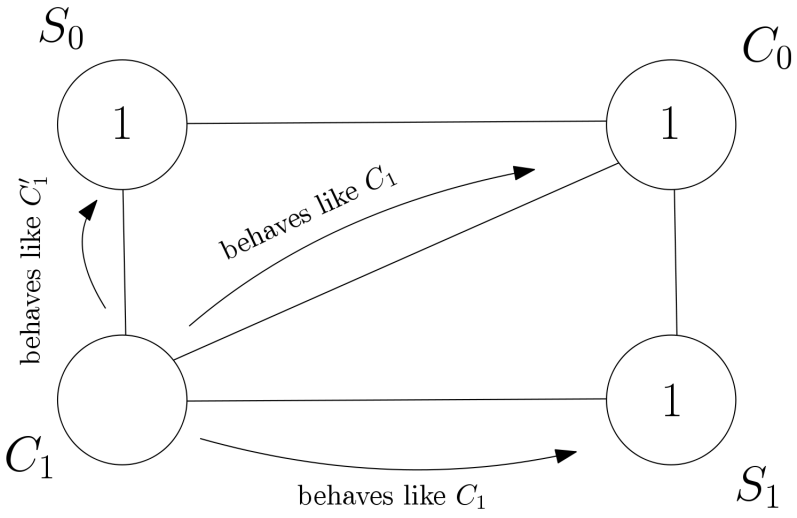


Figure 4: Scenario 3.

## 2 An Algorithm for $f$-Resilient Consensus in a $(2f+1)$-Connected Network

Assume each process knows the topology of the network. A modified version of the exponential algorithm from the textbook is used (section 5.2.4). Before running the exponential algorithm, each process deterministically computes the same set of $2f+1$ node-disjoint paths between every pair of processes in the network. Next, information gathering begins.

Instead of gathering information for the first $f+1$ rounds, information is gathered for $f+1$ phases. A phase consists of $n$ rounds. In the first round of phase 1, each correct process $p_i$ begins sending its input value to all other processes. This is performed as follows. For each process $p_j \neq p_i$, $p_i$ sends a set containing a triple $(i, j, x_i)$ to the first node on the path of each of the $2f+1$ node-disjoint paths connecting $p_i$ and $p_j$ that were determined prior to running the algorithm. If multiple messages must be sent along the same edge, they are sent together in a set. For the next $n-1$ rounds of phase 1, each process $p_i$ receives all messages on its inbuffers, and forwards each message $(j, k, x_j)$ to the next node on the node-disjoint path connecting $p_j$ and $p_k$ that $p_i$ is a part of. If $p_i$ receives a message from a process whose source and target processes don't match with the node-disjoint path that $p_i$ is a part of, $p_i$ does not forward the message.

At the end of round $n$ of phase 1, each correct process $p_i$ counts the maximum number of times $d$ that it received the same input value $x$ from another process $p_j$. If $d \geq f+1$, $p_i$ associates the input $x$ with process $p_j$. Otherwise, $p_i$ associates a default value $\perp$ with $p_j$. The information each process knows is stored in a tree in the same fashion as the original algorithm. In general, in the first round of phase $r$, each correct process sends the $r^{\text{th}}$ level of its tree to all other processes, using the same forwarding method that was used in phase 1.

By Lemma 2, by the end of each phase each correct process successfully identifies the message sent to it by every other correct process at the beginning of that phase. After round $n$ of phase $f+1$, each correct process uses the same recursive majority vote that was used in the exponential algorithm to decide its output. Thus, the modified algorithm behaves in the same way as the exponential algorithm on a complete graph. Correctness follows from the proof of the exponential algorithm.

**Lemma 2.** *In each round, each correct process $p_i$ correctly identifies the message sent by all other correct processes $p_j$ to $p_i$.*

*Proof.* Each correct process $p_j$ that sends some information $v$ to $p_i$ sends it along $2f+1$ node-disjoint paths. Since there are at most $f$ Byzantine processes, there are at least $f+1$ non-faulty paths. Since the length of each non-faulty path is at most $n-1$, by the beginning of round $n$ the messages on all non-faulty node-disjoint paths from $p_j$ to $p_i$ will have arrived to $p_i$, and $p_i$ will have received at least $f+1$ copies of $v$. Thus $p_i$ will identify $v$ as the message sent by $p_j$ that round. □