ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2016 Prof. R. Wattenhofer

Distributed Systems Part II

Solution to Exercise Sheet 3

1 Consensus in a Grid

a) This is the algorithm described in Exercise **2**.

The goal of the algorithm is that every node learns the initial values of all nodes. Each node stores the received values in a set "allValues". Every round, all newly received information is forwarded to all neighbors, until no new information is received anymore. In the first round, every node sends its own value to its neighbors.

Correctness: Let us look at a particular node u. Note that all messages are transported on the shortest path to u. Hence, in round 1, u receives all messages from its direct neighbors. In round 2, all values from nodes in distance 2. And so on. Hence, u receives a new value every round, until it received all values.

Termination: Since the longest path from any node to any other node is an upper bound on the duration until a node receives a value, we know that the algorithm terminates after at most $l + 1$ rounds. (The last new information could be received in round l, and in round $l + 1$ the node realizes that no new values will arrive and terminates.)

Algorithm 1 Simple Consensus in Grid

```
1: allValues = {(myId, myValue)}
2: recv = {(myId, myValue)}
3: for Round 1 to ∞ do
4:    Send values(recv) to all neighbors
5:    recv = receive tuples from neighbors
6:    remove all tuples from recv which are already in allValues
7:    if recv = ∅ then
8:       No new tuple received
9:       return minimum value in allValues
10:   else
11:      allValues = allValues ∪ recv
12:   end if
13: end for
```

b) We already showed that $l + 1$ is an upper bound on the runtime in **a)**. In fact, the nodes on the corners of the grid require exactly l time until they learn the value of the corner on the opposite side. Since it requires one additional round to realize that this was the last time a new message arrived, the runtime is exactly $l + 1 = w + h + 1$.

c) If $w \approx h$ one can arrange the faulty nodes in two diagonals (see Figure 1). The longest shortest path has a length of $l \approx 3 \cdot (w + h)$. For every w and h it is possible to arrange the faulty nodes in a pattern as shown in Figure 2. In that case, a longest shortest path of $l \approx 2 \cdot (w + h) + c$, where c is a small constant, can be achieved.

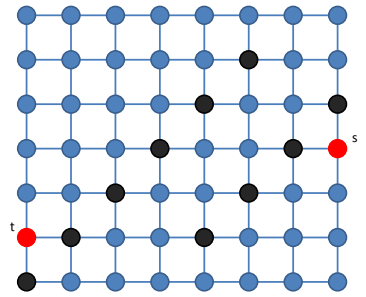


Figure 1: Strategy for $w \approx h$ with faulty nodes marked as black. The longest shortest path l leads from t to s

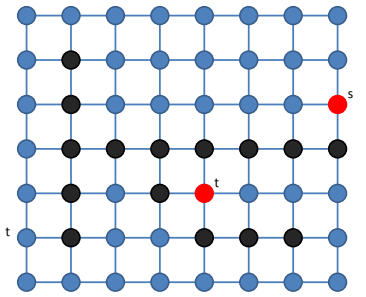


Figure 2: Strategy for any grid that achieves a longest shortest path of $l \approx 2 \cdot (w + h)$

d) One byzantine node, placed next to a corner. In that case, the corner node has one byzantine neighbor, and one correct neighbor. If the byzantine node pretends to send messages in such a way, that it looks like a completely normal execution of the algorithm, but with wrong initial values, the corner node gets two different pictures, and cannot determine which one is right. Any algorithm must choose one of the nodes to be byzantine and listen to the other, and since there is no good way to do that, any algorithm for the corner node will violate the agreement property in at least 50% of the executions.

2 Consensus in a Grid: The Algorithm

a) Let us choose any node u. We show that u receives the value of every running node before it terminates.

Recall that the graph is connected, and that the nodes which are farthest away from u are k hops away, with $k \leq l$. Thus, by definition of the distance, there is a shortest path from

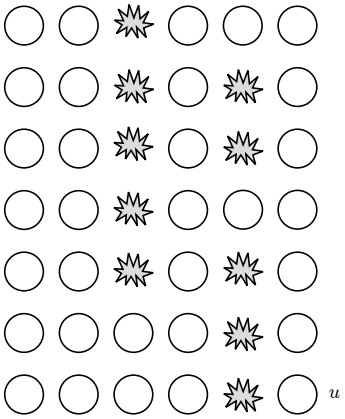
each of those nodes to u of length k. Observe that on that path there is exactly one node u_1 with distance 1 to u, one node u_2 with distance 2, ... up to the last node u_k with distance k. As u hears the values of all nodes with distance i to u in round i, it follows that u receives a information about a new node in each round $1, 2, \ldots, k$. Since k is the maximal distance of any node to u, it follows that u does not terminate before it heard a message from all nodes.

Since the argument applies to any node u, it holds for all nodes, concluding the proof.

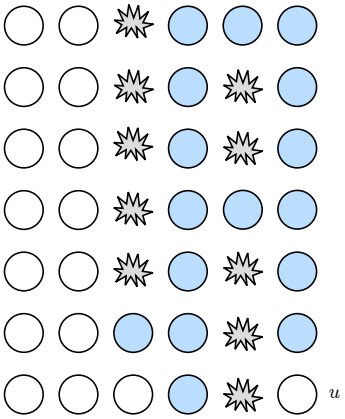
b) A node u hears a message from v in round $d(u, v)$. Thus, it has heard a message from all nodes at latest in round l. Hence u can impossibly hear any new information in round $l + 1$, and will therefore terminate.

Thus every node terminates at latest after $l + 1$ rounds.

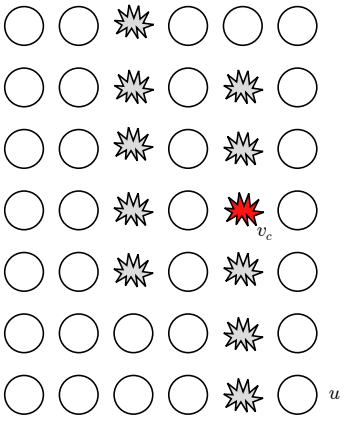
c) We show that a node u in the bottom right corner of the grid terminates too early:



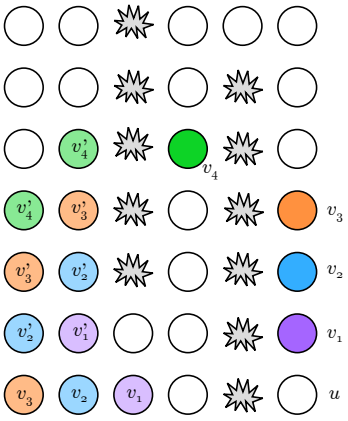
(a) The initial configuration of the network. Grey nodes are crashed initially.



(b) After 8 rounds, node u knows the values of all blue marked nodes...



(a) ... and then node v_c crashes!



(b) Which nodes close to u know the values of which nodes already? We marked nodes v_i and the respective v_i' of which they already learned the initial values.

After 8 rounds, the v_4 is the closest node to u which knows about v_4' or values which are even farther away from u. In Round 9, u learns new values v_1', which are now stored by v_1. In rounds 10 and 11 it learns the values now stored by v_2, respectively v_3.

However, in Round 12, u does not learn a new value! The reason is that the initial values of v_4' have to take a longer path to reach u (see Figure 5)! Thus, in Round 12, u does not learn

a new value and terminates. Since u has not learned the values from v_4' (and others), we have shown that there are executions (and certain failure patterns) in which certain nodes terminate too early.

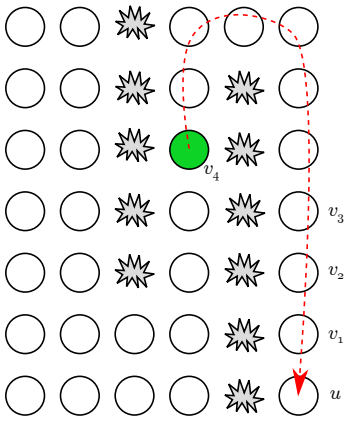


Figure 5: Values routed via v_4 take a longer time to reach u.

3 Revisiting Paxos

In our execution A and B contact two different majorities which only overlap in one node: The byzantine node u.

First A contacts all nodes with ticket number 1. A only receives messages from the red majority, all messages from all other nodes are slow/lost/... Since A received a ticket from a majority, A successfully executes the entire steps of Paxos, with command 1. Hence, A as well as the red majority agree that command 1 is chosen.

Afterwards B contacts all nodes with ticket number 2. Unfortunately B only receives messages from the blue majority, all messages from other nodes are slow/lost/... Therefore, the B receives messages from a majority, but the only node within that majority that knows that a command was already chosen is u! Thus B learns from u that a command was stored, but since u experiences a byzantine fault in its memory cell C, u answered with `ok`(1, 0). So B learns – incorrectly! – that command 0 was agreed upon, and proposes 0. Since the ticket number is valid, all blue nodes accept 0 as the decision, and the system failed to agree on one value.

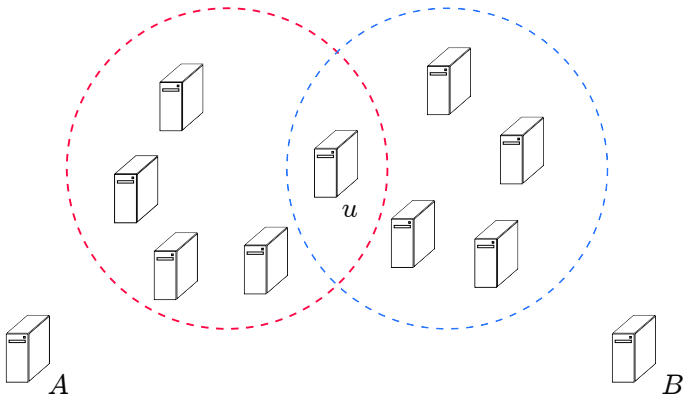


Figure 6: An unfortunate case with a simple byzantine node can lead to disagreement!

4 Consensus in a Grid. . . again!

a) Our algorithm is very simple: The node in the center is the leader, and it sends its value to all neighbors. All other neighbors relay the value to all neighbors and terminate.

Since w and h are even (recall that w, h denote the number of *edges* and not nodes) the center is well defined; and since the center knows its coordinates and w and h, it can initiate the broadcast. Analogous to exercises **1** and **2** we know that the runtime of the algorithm is the distance from the center to the farthest away node, which is the corner. Hence, the runtime is exactly $(w + h)/2$.

b) We study three different possible distributions of initial values:

All Zero All nodes start with initial value 0.

All One All nodes start with initial value 1.

Half-Half The bottom left half (including the "middle diagonal") starts with 0, the top right quarter (without the "middle diagonal") starts with 1.

The "middle diagonal" is defined by all nodes which have distance $(w + h)/2$ from the lower left corner, respectively the upper right corner. Without loss of generality, assume that $w \leq h$ (otherwise, simply turn the grid by 90°). The coordinates of these nodes are $(i, (w + h)/2 - i)$, for $i \in [0, w]$. The diagonal includes the center of the grid and is shown below.

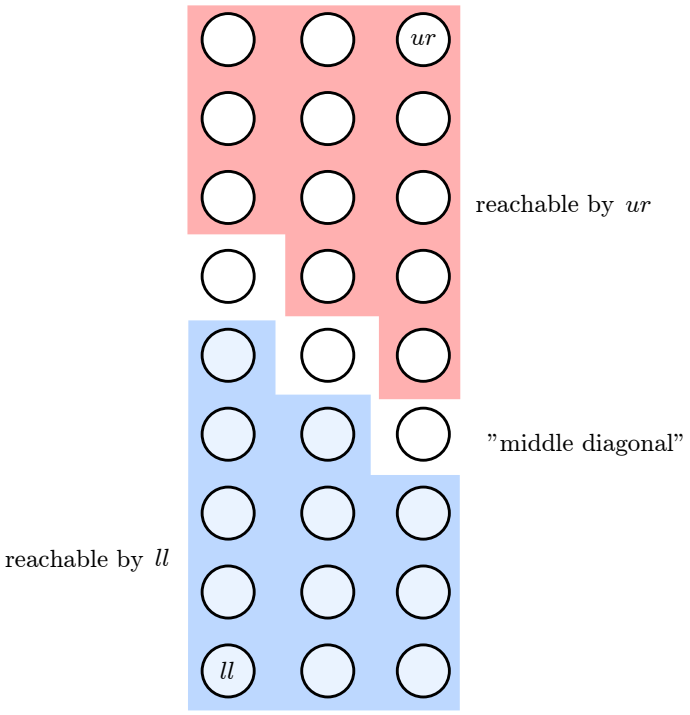


Figure 7: A grid with $w = 2$ and $h = 8$.

Assume that there is an algorithm which terminates after $(w+h)/2-1$ many rounds. Observe that after r rounds, any algorithm can only forward information up to a distance of r. Thus, neither the lower left corner, nor the upper left corner will receive any information from the diagonal or the nodes on the opposing side of the diagonal, since these nodes are at least in distance $(w + h)/2$.

Hence we study the information received by the lower left corner ll and the upper right corner ur in all three scenarios:

	ll	ur
All Zero	only 0 values	only 0 values
All One	only 1 values	only 1 values
Half-Half	only 0 values	only 1 values

Recall that the validity property of consensus requires the decision value to be one of the initial values. Thus, ll and ur must decide for 0 in the scenario "All Zero", and 1 in the scenario "All One". Since ll cannot distinguish "All Zero" and "Half-Half", and we require a deterministic algorithm, ll will decide for 0. However, with an analogous argument, ur cannot distinguish "All One" and "Half-Half" and must therefore decide for 1.

Hence, if the validity property is satisfied in both "All Zero" and "All One", the agreement property will be violated in the scenario "Half-Half". Thus, no deterministic algorithm with runtime less than $(w + h)/2$ can exist.