Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2011 Prof. R. Wattenhofer / B. Keller

Distributed Systems Part II

Solution to Exercise Sheet 2

1 Consensus with the Aid of a Wall

a) The Algorithm looks like this:

```
Choose the color of the place you want to meet.
Go to the Painter and instruct him to paint the wall in the corresponding color
Look at the “before and after” picture which you get from the painter.
if {the wall was white before} {
  the meeting place is the one you have chosen
} else {
  the meeting place is the place according to the “before color”
}
```

b) No. What Alice and Bob can do, is in a sense the same as the RMW-primitive swap. As swap is overwriting its consensus number is two. So there is no way to ensure that more than two persons can meet at the same place.

c) If the wall is in front of the painters' shop it is basically the same as the RMW-primitive Compare and Swap (because you would see if the painter is already painting, or if someone is in the shop and instructing the painter to paint) which has consensus number ∞. This means, that infinitely many persons could meet. The Algorithm would look like this:

```
Choose the color of the place you want to meet.
Go to the Painter
Look at the wall in front of the painters' shop
if {the wall is white } {
  enter the shop and instruct the painter to paint the wall in your color
  the meeting place is the one you have chosen
} else {
  the meeting place is the place according to the color of the wall
}
```

2 Consensus through “Fetch and Multiply”

I would tell him, that this is not possible. The method “Fetch and Multiply” he uses is commutative, therefore the consensus number of his algorithm cannot exceed two.

3 Consensus with Byzantine Failures

No 2-resielient algorithm for 6 processes without authentication exists.
This is a proof by contradiction: It is known that consensus for 3 processes, one of them Byzantine, cannot be solved. This proof can be found in the lecture notes on slides 108ff. Assume now that consensus for 6 processes could be solved by an algorithm a. We create a new algorithm b which solves consensus for 3 processes using a. The algorithm b works as follows: each of the 3 processes (hosts) simulates 2 processes (guests). The input of the guests is equal to the input of their hosts. Correct hosts simulate correct guests, the Byzantine host simulates Byzantine guests. The 6 guests solve consensus using a. The hosts copy the decision of their guests (all guests make the same decision). This is a contradiction! Since b works if a works, therefore a must be wrong (as it is known, that b cannot exist.) Hence there exists no algorithm a solving consensus for 6 processes.

4 Consensus in a Grid

a) The goal of this protocol is for every process to know the input of all the other processes. Each process internally allocates an array of size $w\cdot$, each entry of the array represents the input of one process and initially is set to '?'. The process fills the array until no '?' are left, then the process can make its decision (e.g. the minimal value received). Whenever a process gets new information, it sends its updated array to all its four neighbors. At the beginning the "new information" is its own input.

b) The number of required rounds is the length of the longest shortest path between any two processes. In the case of a grid this is $w + h$.

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 $\approx 3 \cdot (w + h)$. So the number of required rounds is about $3 \cdot (w + h)$. In each case it is possible to arrange the faulty nodes in a pattern like in figure 2. So a longest shortest path of $2 \cdot (w + h) + c$ where c is a small constant can be achieved.

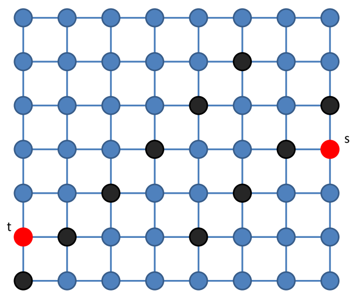

Figure 1: Strategy for $w \approx h$ with faulty nodes marked as black. The longest shortest path leads from t to s

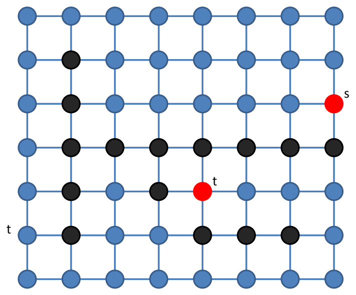

Figure 2: Strategy which can be done in every grid and achieves a longest shortest path of $\approx 2 \cdot (w + h)$

d) Byzantine processes can catch and modify messages. In a setting like in figure 3 only one of four messages reaching t is correct. Process t cannot find out which messages are wrong, thus there is a high probability for t to make false decisions.

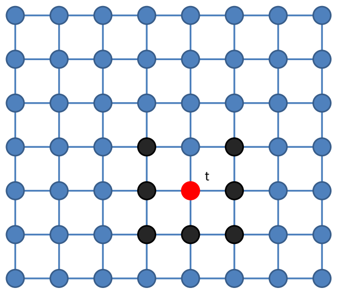

Figure 3: Byzantine processes (dark) can "bombard" process t with wrong information and lead t to a wrong decision

5 Consensus in a Hypercube

a) We only need to alter how broadcast is implemented, all other parts of the king algorithm remain unchanged.

In a hypercube broadcast is not as easy as in a fully connected network. Processes need to forward the messages, and any byzantine process can alter the messages it has to forward.

We can increase the chance for a message to reach its destination unaltered by sending the message over multiple paths. To be more exact: in a hypercube with dimension m we can always find m non-intersecting paths between two processes.

A process may receive the same message up to m times, but the faulty processes could have altered some of them. The process needs to decide what the original message was. Since there is no authentication available, the process can do nothing but assume the message that is most often received is the original message. This leads to $f < m/2$.

b) The king algorithm has $f + 1$ phases, in each phase the algorithm sends three broadcast. So the algorithm requires $3 * (f + 1)$ times the duration of a broadcast.

How much time does a broadcast require? Approximately m rounds, because the longest shortest path leading from one corner of the hypercube to the opposite corner has length m.

To be more exact, a broadcast requires $m + 1$ rounds: If process a sends to process b, then b must receive the message up to m times. In a hypercube every process has coordinates which we can express as binary strings of length m. A path between a and b can be seen as sequence of bits that are switched. Example: if $c_a = 000$ and $c_b = 111$ then one path from a to b first visits the processes at 001 (switching bit 3), then the process at 011 (switching bit 2), and finally b (switching bit 1).

The length k of the shortest path between a and b is the number of bits c_a and c_b differ in. There are k non-intersecting paths of length k between a and b. (The i'th path would start with switching the i'th different bit, then the $i + 1$'th different bit, etc.). This leaves $m - k$ paths with a length greater than k.

These $m - k$ paths need first to switch a bit which is the same in c_a and c_b (otherwise the path would intersect with one of the shortest paths). This bit needs to be switched back at the end of a path. Other than that a path needs to switch k bits that differ, leading to $k + 2$ operations all together.

We set k as big as possible, but ensure $k < m$. This means we set $k = m - 1$ and find the longest path a message must ever travel has length $m + 1$.