Illustration of Our Project

Description

For a set of nodes, each have a value which may not always be the same as others at first. Our goal is to let every node in the set to have the same value as any other different nodes after contacting a period of time mutually. (During the contact, some nodes may go wrong and be incapable of sending messages to the rest).

Algorithm

Each node in the network consists of two sets, one is defined as New-Value which only contains one element v (the original value) at first, while and the other is defined as Old-Value, an empty set at the very beginning. So structure <old-values | new-values> is defined to tag the state of a node and to store its value.

For every node executes the following steps:

1. Send all the values in New-Value to its neighbors.
2. Union the New-Value and Old-Value (get a bigger set Old-Value)
3. Insert the value receiving from other nodes to a new set New-Value' if it hasn' t been received before else ignore it. Let New-Value' be the New-Value in the next iteration.
4. If one of the termination conditions has been met, then the algorithm is over and every node can receive the same minimum. If not, go back to Step 1.

The algorithm is over when arbitrary termination conditions are met after one turn of iteration, else go back to Step 1 and start a new iteration.

When to stop iterating?

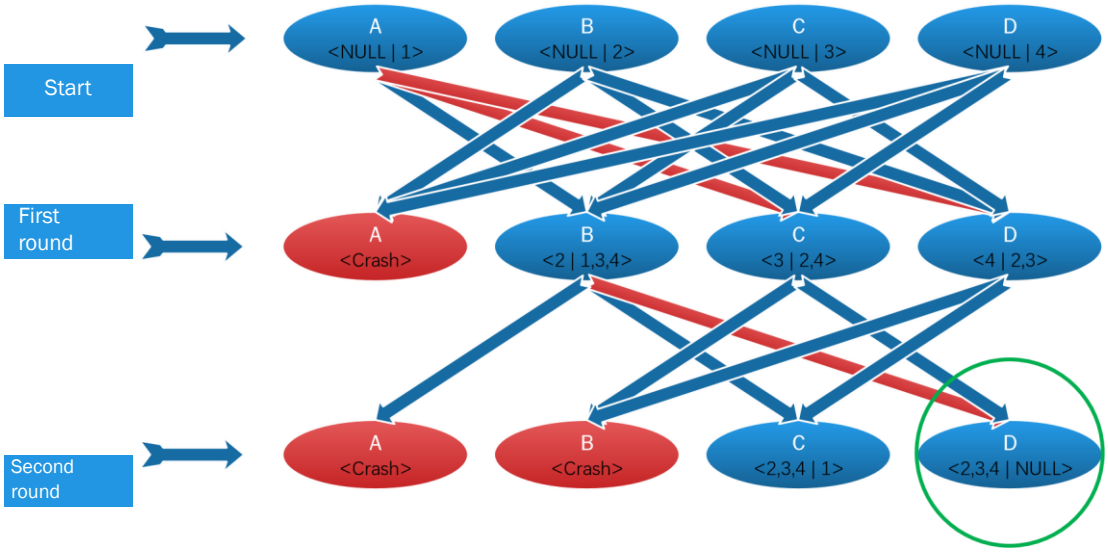
Hypothesis: When a node doesn' t receive any value which it hasn' t received yet any more, it stops iterating over the program.

The hypothesis can be proofed to be wrong by the counterexample below.

Suppose there are four nodes A B C D initialized with 1 2 3 4.

The procedure is showed in the picture below. As the picture depicts, the color of nodes represents working state, red means the node has crashed and blue means the node is working normally. The color of edges

represents sending state. red means failure or other accidents occur when sending the values and blue means the values can be sent successfully.



In the first iteration:

As is showed in the picture that A was going to send all its new values (though there is only 1) to BCD, but unfortunately it crashed sometime during the sending and only B received the message successfully; likely, B sent 2 to A, C, D; C sent 3 to A, B, D; and D sent 4 to A, B, C but luckily no more crash occurred. and the process can be described as follow:

A<NULL | 1> —> B，then crash

B<NULL | 2> —>A、C、D

C<NULL | 3> —>A、B、D

D<NULL | 4> —>A、B、C

By the end if the first iteration, the state of each node changed to A<Crash>、B<2 | 1,3,4>、C<3 | 2,4>、D<4 | 2,3>

In the second iteration:

All the nodes “survive” the first iteration(BCD), sent its value to the nodes which sent it a message in the previous iteration and only B crashed sometime during the sending.

B<2 | 1,3,4> —> A、C，then crash

C<3 | 2,4> —> B、D

D<4 | 2,3> —> B、C

Current state of each node changed to A<Crash>、B<Crash>、

C<2,3,4 | 1>、D<2,3,4 | NULL>

Note that by the end of the second iteration, D didn’ t receive any new value, so it stopped iterating as supposed, and the final value of D is 2. However, in next iteration, since no value was received by C, C would stop its iteration with final value 1, which was different from that of D, and contract occurred.

Let us go further, suppose that the nodes stop iterating when it doesn’ t receive any message in some turn, and we can easily find contracts if we go through the procedure above with more nodes.

As we can see, if there exits some iteration during which no node crashed, then all nodes can reach a consensus. Therefore, we draw the conclusion that only when no node crashes during the current iteration and there is no more new-values received, can the nodes stop processing the algorithm.

So the key of knowing when to stop iterating is to find a iteration during which no node crashed And it can be realized by adding an extra sign to every message sent to show that the node sent the message is still operating well, thus the change of the number of node between the two rounds can be detected and whether there is node crashed can be known.

Explanations (for code)

We defined a class, Node, to represent each node and initial it with four parameters including with index (the identity number of the node in the whole network), value (the original value of the node), crash(indicates in which round the node goes wrong) and all node (information about all nodes’ IP port in the network)

```
class Node:
    def __init__(self, index, value, crash, allnode):
        self.skt = socket.socket(socket.AF_INET, socket.SOCK_DGRAM);
        self.skt.bind(allnode[index])
        self.skt.settimeout(0.5)

        self.index = index
        self.crash = crash
        self.round = 1
        self.nodes = allnode
        self.values = {value : "new", -1 : "new"}
        self.wait = 0
        self.last = 0
```

There are three functions in the class Node:

1. send_news which sends its own new messages to others
2. recv_news which is responsible of receiving messages from other nodes and
3. get_same which executes both the send_news and recv_news iteratively.

Booting a node is realized by creating a thread using the function get_same.

```
node1 = Node(0, 1, 1, nodes)
node2 = Node(1, 2, 2, nodes)
node3 = Node(2, 3, 3, nodes)
node4 = Node(3, 4, -1, nodes)
node5 = Node(4, 5, -1, nodes)
node6 = Node(5, 6, -1, nodes)

print("begin")
Thread(target=node1.get_same).start()
Thread(target=node2.get_same).start()
Thread(target=node3.get_same).start()
Thread(target=node4.get_same).start()
Thread(target=node5.get_same).start()
Thread(target=node6.get_same).start()
```

There are six nodes in the network, and all of them are started by the method described above.

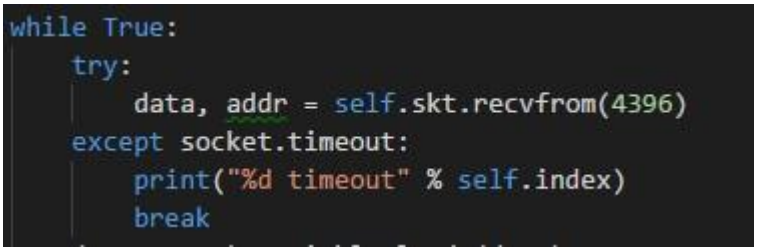
In the network with six nodes (0-5), let the case to be that the first node Crashes in the first round the second in the second round and the third in the third round, and we can get the following result.

```
4 last Value: 1, round 5
3 timeout
3 last Value: 1, round 5
5 timeout
5 last Value: 1, round 5
```

According to the picture above, it can be seen that the rest three nodes 3,4,5 which operate properly have been synchronized by the fifth round. In fact, they have reached an agreement during the fourth round while terminated in the fifth round.

The following articles will show how we program to simulate the algorithm above in the synchronous network environment

As only when the overtime signal occurred, can the block of recv function be stopped during the communication of Socket, we realize the function as follow:



```
while True:
    try:
        data, addr = self.skt.recvfrom(4396)
    except socket.timeout:
        print("%d timeout" % self.index)
        break
```

Make every node to wait for a second after finishing running the send_new and the recv_new function every round to ensure that all nodes have completed the current iteration (similar to a unified clock)