Fault-Tolerance in Asynchronous Networks – Probabilistic Consensus
Impossibility of Consensus

- We know there exists an infinite execution in any consensus algorithm for asynchronous networks that tolerates $\geq \text{failures}$.

- Why exactly is such an algorithm not a consensus algorithm?
Definition: T-crash fair executions

A t-crash-robust algorithm is a consensus algorithm if it satisfies:

- **Termination**
  - All correct processes *eventually* decide

- **Agreement**
  - In any configuration, the decided processes should have decided for the same value (0 or 1)

- **Non-triviality**
  - There exists at least one possible input configuration where the decision is 0
  - There exists at least one possible input configuration where the decision is 1
    - Example, maybe input “0,0,1”->0 while “0,1,1”->1
A t-crash-robust algorithm is a consensus algorithm if it satisfies:

- **Termination**
  - All correct processes *eventually* decide

- **Agreement**
  - In any configuration, the decided processes should have decided for the same value (0 or 1)

- **Non-triviality**
  - There exists at least one possible input configuration where the decision is 0
  - There exists at least one possible input configuration where the decision is 1
  - Example, maybe input “0,0,1”→0 while “0,1,1”→1
How to make it possible

- Let's make an algorithm that makes such infinite executions unlikely!

- Even if such infinite executions would be unlikely, this would not formally be a consensus algorithm

- How to change the definition?
  - What do we require for “probabilistic consensus” algorithms?
Probabilistic Consensus

- A probabilistic t-crash-robust algorithm is a consensus algorithm if it satisfies:

  - **Termination/Convergence**
    - \( \lim_{k \to \infty} P[\text{a correct process has not decided after } k \text{ steps}] = 0 \)

  - **Agreement**
    - In any configuration, the decided processes should have decided for the same value (0 or 1)

  - **Non-triviality**
    - There exists at least one possible input configuration where the decision is 0
    - There exists at least one possible input configuration where the decision is 1
      - Example, maybe input “0,0,1”\(\to 0\) while “0,1,1”\(\to 1\)
Bracha & Tuogeg Probabilistic Consensus

- Can tolerate $t < N/2$ crash-failures
  - $N=10$ $t=4$, $N=11$ $t=5$

- Cannot wait on more than $N-t$ receives

```plaintext
var
  value_p : (0,1)  init x_p
  msgs_p[0..1] : int  init 0

begin
  while y_p=b do
    begin
      msgs_p[0]:=msgs_p[1]:=0;
      shout <vote, value_p>;
      while msgs_p[0]+msgs_p[1] < N-t do
        begin
          receive <vote, v>;
          msgs_p[v]++;
        end
      end
      if msgs_p[0]>msgs_p[1] then value_p:=0
      else value_p:=1;
    end
end
```

- My initial value
- Two counters (vector), both initially 0
- Keep looping until decision
- Reset both counters to 0
- Broadcast my value and my round number
- Count and receive $N-t$ votes
- Choose majority
Bracha & Tuoeg Probabilistic Consensus

- **Asynchronism:**
  - Shout and receive should go in *rounds*
  - Avoid messages being consumed in the *wrong* round

```plaintext
var
  value_p : (0, 1)  init x_p
  round_p : int     init 0
  msgs_p[0..1] : int init 0

begin
  while y_p = b do
    begin
      msgs_p[0] := msgs_p[1] := 0;
      shout <vote, round_p, value_p>;
      while msgs_p[0] + msgs_p[1] < N-t do
        begin
          receive <vote, r, v>;
          if r > round_p then
            send <vote, r, v> to p;
          else if r = round_p then
            begin
              msgs_p[v]++;
            end
        end
      end
      if msgs_p[0] > msgs_p[1] then value_p := 0
      else value_p := 1;
      round_p++;
    end
  end
```
Absolute Majority

- **Idea:**
  - If any node sees more than N/2 similar values, it has *witnessed* an *absolute majority*

- Each time you vote V, announce the number of V’s you have seen
  - I.e. attach the *weight* of your vote

- Other nodes can tell from your weight if you are a witness or not

- The vote of a witness overrides any other votes

- If more than t witnesses are seen in one round, *finally decide!*
**Idea:** If any node sees more than N/2 similar values, it has *witnessed* an absolute majority.
Agreement
- If any process decides, then all processes decide in the next two rounds

Proof:
- First notice that in one round, no two processes can witness for different values (0/1)
  - I.e, for two witnesses $p$ and $q$:
    - $p$ receives $> \frac{N}{2}$ ones, and $q$ receives $> \frac{N}{2}$
- We know processes decide if they see $t$ witnesses
  - If $p$ decides 1 in round $k$, there were $> t$ witnesses in the round
  - So everyone in round $k$ has seen at least one witness for 1
  - Everyone will therefore vote 1 in round $k+1$
  - In round $k+2$, everyone will have seen $t$ witnesses, and hence will decide!
Convergence?

- How do we know we will ever see a first decide?
  - Termination/Convergence
    - $\lim_{k \to \infty} P[\text{a correct process has not decided after } k \text{ steps}] = 0$

- Assume fair scheduling
  - In any round, for any two nodes $p$ and $q$, $p$ will receive a message from $q$ with positive probability
  - For a subset $S$ of correct processes $N-t$, there exists some positive probability $p$ (maybe very small) that every process in $S$ receives the votes of the processes in $S$ in some round $K$
    - With probability $p^3$ this happens in the next two rounds also
    - Hence in round $K$, processes in $S$ receives the same vote and vote similarly in round $K+1$
    - In round $K+1$, there will exist $N-t$ similar votes, hence someone in $S$ have seen a witness in the next round (decide)
Summary

- We have shown that probabilistic consensus algorithms exist for the asynchronous networks.
- It can tolerate up to approx. half of the processes failing!