Distributed Systems
COMP 212

Lecture 20

Othon Michail

UNIVERSITY OF LIVERPOOL

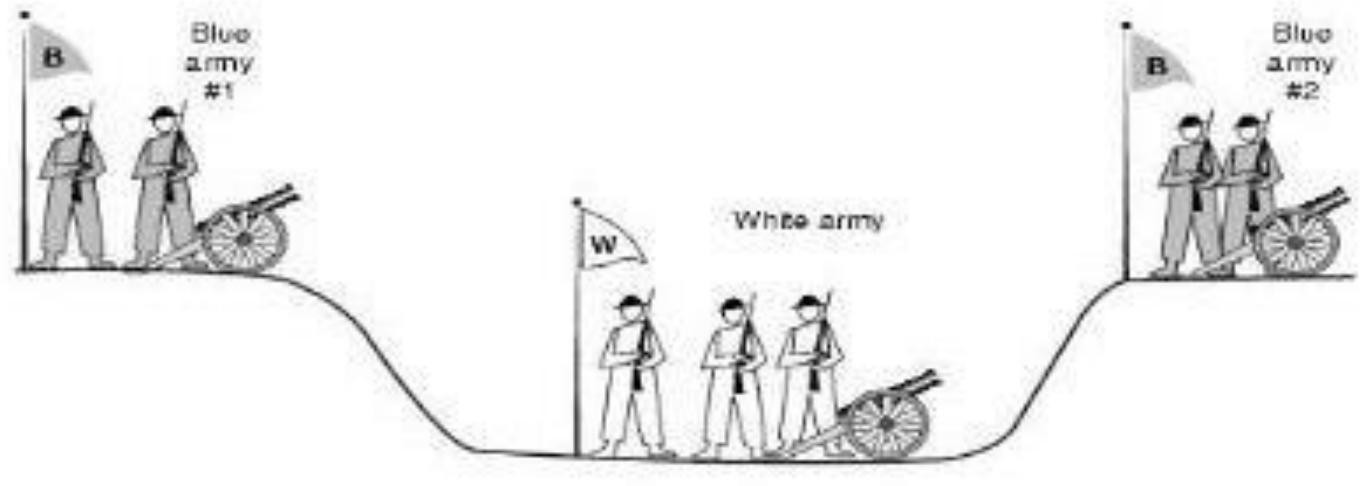
Distributed Consensus with Link Failures

Agreement with Link Failures

- Each processor in the network
  - starts with an initial value of a type
  - must eventually output a value of the same type
- Goal: All processors must agree on their output
- Examples:
  - Input: Measurement of component condition, Output: Agree if the component is faulty
  - Input: Opinion about whether a transaction ought to be committed or aborted, Output: Choice on whether to commit or abort
- Easy to solve if no failures exist
- *What if messages can be lost?*

The Coordinated Attack Problem

- “Two blue armies need to simultaneously attack the white army to win; otherwise they will be defeated. The blue army can communicate only across the area controlled by the white army which can intercept the messengers.”



- *Is there a solution?*

Formalising the Problem

- *n* processors (*n* attacking armies in general)
- In any connected undirected network
- Each processor knows the entire network (complete knowledge)
- *input*(*u*) ∈ {0, 1}, e.g.,
  - ‘1’: “attack” or “commit transaction”
  - ‘0’: “do not attack” or “abort transaction”
- Synchronous communication, but now
- Any number of messages may be lost during the execution

Formalising the Problem

- Goal: Each processor u will eventually give an $output(u) \in \{0, 1\}$
- Outputs must satisfy the following conditions:

Agreement: No two processors decide on different values.

Validity:

1. If all processors start with 0, then 0 is the only possible decision value.
2. If all processors start with 1 and all messages are delivered, then 1 is the only possible decision value.

Termination: All processors eventually decide.

Impossibility of Coordinated Attack

- Even with non-faulty processes, agreement between even two processes is not possible in the face of unreliable communication

Proof. Processes *P1* and *P2.* Any number of messages can get lost.

- Both 0: Assume they both have input 0. If no message is lost then they eventually both agree on output *NO*.
- By a chain argument, losing the last message, then the message before that, … we can prove that *P1* and *P2* also have to agree on *NO* even when all messages are lost
- Both 1: Similarly, when they both have input 1, they have to agree on *YES* even when all messages are lost
- But then, when *P1* has input 1 and *P2* has input 0 and all messages are lost, *P1* has to output *YES* while *P2* has to output *NO*

➢ (=>) If they agree on the *both*-0 and *both*-1 cases, then they cannot agree on the *mixed* 0-1 case. □

Distributed Consensus with Processor Failures

Agreement with Processor Failures

- Same setting, but now processors may fail instead of links
- In particular, processors may only crash/stop
- Each processor in the network
  - starts with an initial value from a set X
- Non-faulty processors
  - must eventually output a value from X
- Goal: All non-faulty processors must agree on their output

Crash/Stop Failures

- At any point during execution, a processor might simply stop
- Might stop in the middle of a message-sending step
  - only a subset of messages might be sent
  - we assume that *any* subset might be sent
- Might stop after sending its messages but before processing its incoming messages

Correctness Conditions for Agreement

- Outputs must satisfy the following conditions:

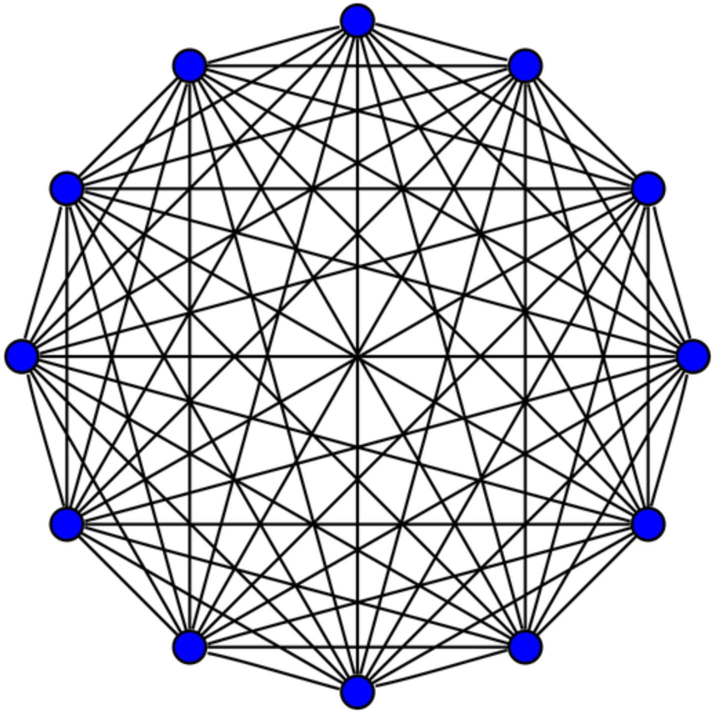
Agreement: No two processors decide on different values.

Validity: If all processors start with the same initial value $s \in X$, then s is the only possible decision value.

Termination: All non-faulty processors eventually decide.

Setting

- n processors
- Organised in a complete network
  - All-to-all in both directions
- Goal: Solve the agreement problem in this setting



- Conventions:
  - At most f processors may crash and the upper bound f is known to the algorithm
  - s_0: A prespecified default value from set X
  - b: An upper bound on the #bits needed to represent any single value in X

Solution: General Idea

Algorithm FloodSet

- Simple algorithm
  - All processors just forward to all, every value from *X* that they have heard and
  - Apply a simple decision rule in the end
- Idea: If in any round no failure occurs, then it is guaranteed that all processors will "sync" and will maintain the same info until the end

Remark: For simplicity of presentation we here assume that in a round, processors first transmit, then receive and update and then the round ends

Solution: Informal description

Algorithm FloodSet

- Each processor maintains a variable W containing a subset of X
  - initially only the processor's initial value
- For each of $f + 1$ rounds
  - Each processor sends W to all other processors
  - Then adds all the elements of the received sets to W
- After $f + 1$ rounds each processor applies the following decision rule:
  - If W is a singleton set, then decide on the unique element of W
  - Otherwise, decide on the default value s_0

Solution: Pseudocode

Algorithm FloodSet
Code for processor u_i , i ∈ {1, 2,… , n}:

Initially:
u_i knows its $input(u_i)$ and a default value $s_0 \in X$
$W_i := \{input(u_i)\}$
$decision_i :=$ '?'
Also has access to the current round and knows the upper bound f on #faults

```
if round ≤ f +1 then
   // The following to be always executed by all processors, i.e.,
   // also in round 1 in which no message has been received
   send ⟨W_i⟩ to all processors              // in one step to all, as the network is complete

   upon receiving ⟨inW_j⟩s from in-neighbours   // one or more W sets arriving from all active processors
      W_i := W_i ∪_j inW_j                     // remember the union of known and received

if round = f + 1 then
      if |W_i| = 1 then                  // if W is singleton
         decision_i := s, where W_i = {s}      // output the unique element of W
      else                               // W contains at least 2 elements
         decision_i := s_0                     // in this case decide on the default value
```

Correctness and Complexity

- Correctness:
  - Show that FloodSet solves the agreement problem in any execution with at most f crash failures
- Time complexity:
  - $f + 1$ rounds (or $f + 2$ depending on the model)
- Communication complexity:
  - size of messages: $O(nb)$
    - b: encoding in bits of the maximum value in X
  - $O((f + 1)n^2)$ messages

FloodSet Correctness

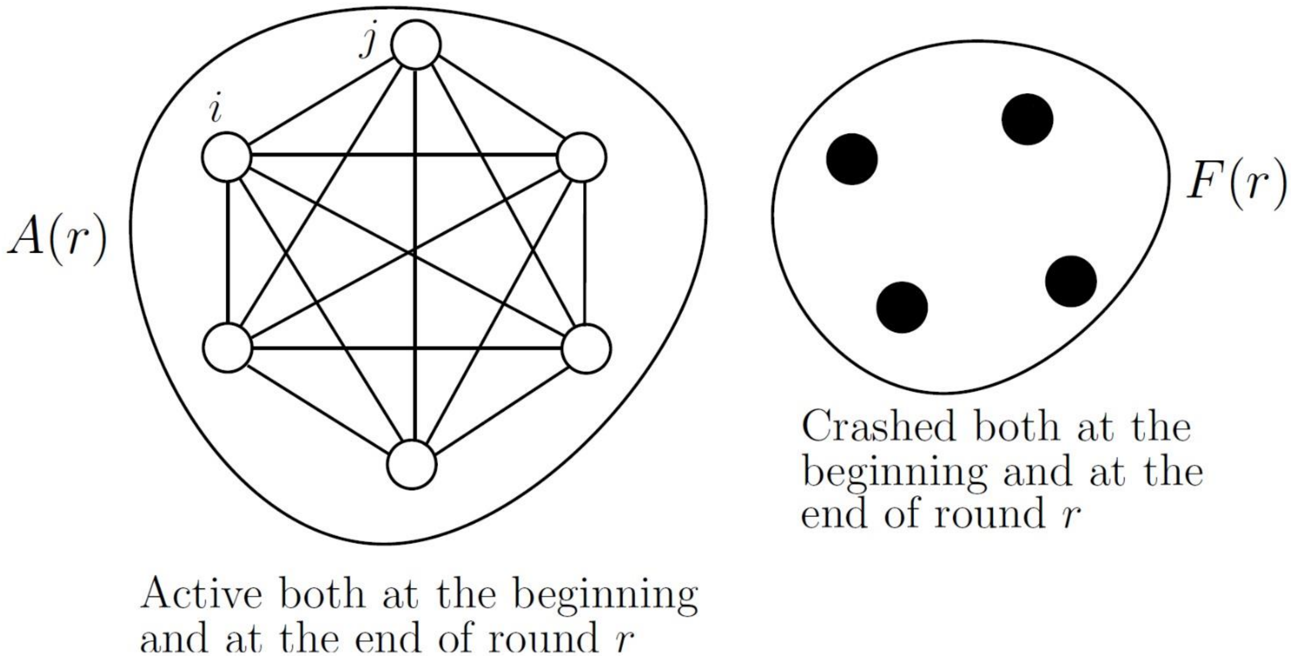
- We have to show that in any execution with at most f crash failures:
  - FloodSet satisfies the agreement, validity, and termination conditions
- Some useful definitions:
  - $W_i(r)$: value of variable W at processor u_i after r rounds
  - Call a processor *active* after r rounds if it hasn't failed by the end of r rounds

FloodSet Correctness

- We first show the straightforward fact that if in some round *r* no active processor fails, then by the end of round *r* all active know the same *W*

Lemma 1. If no processor fails during a particular round r, $1 \leq r \leq f + 1$, then $W_i(r) = W_j(r)$ for all u_i and u_j that are active after r rounds.

Proof.



- $W_i(r) = \bigcup_{j:\, j \in A(r),\text{ including } i} W_j(r - 1)$, for all $i \in A(r)$
- i.e., all the same

□

FloodSet Correctness

- We next prove that if all the active processors have the same *W* sets after some particular round *r*, then the same is true in all subsequent rounds

Lemma 2. Suppose that $W_i(r) = W_j(r)$ for all u_i and u_j in the set $A(r)$ (of processors active throughout round r). Then the same must hold for all rounds r', where $r \le r' \le f + 1$.

Proof.

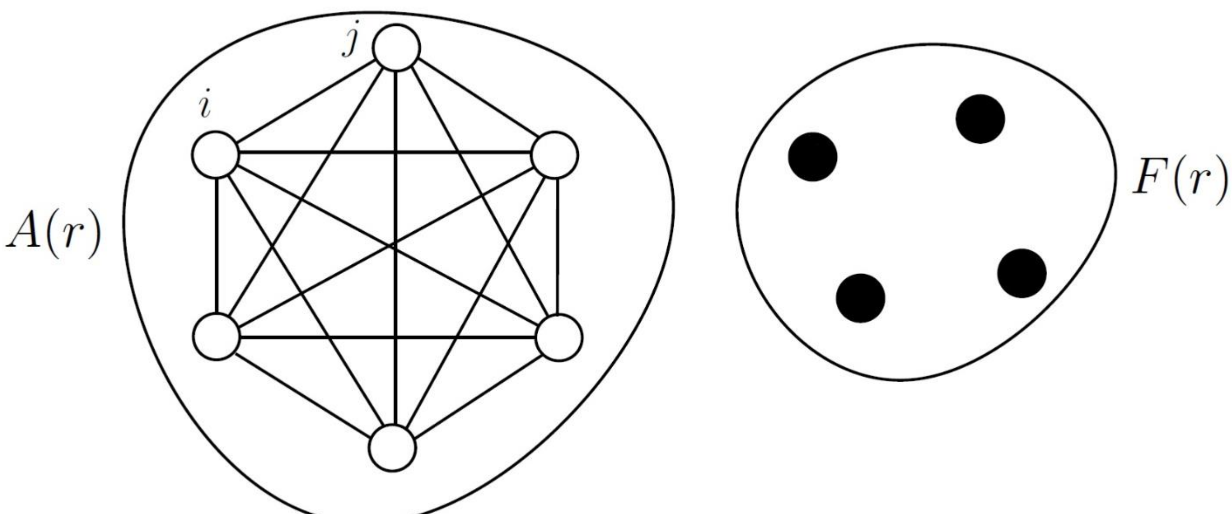
- All processors that haven't failed after *r* rounds have sent their value
- All processors that haven't failed after *r* rounds have identical *W* lists
- No active processor from round *r* and onwards can bring in a new value

FloodSet Correctness

Lemma 2. Suppose that $W_i(r) = W_j(r)$ for all u_i and u_j in the set A(r) (of processes active throughout round r). Then the same must hold for all rounds r', where $r \leq r' \leq f + 1$.

Proof (continued).

- This could only hold if a new info would come in from $F(r)$ (processors already failed), which is impossible
- Thus, W_i, for all processors u_i, will not change in subsequent rounds $\square$



FloodSet Correctness

- The following lemma is crucial for the agreement property

Lemma 3. If processors u_i and u_j are both active after $f + 1$ rounds, then $W_i = W_j$ at the end of round $f + 1$.

Proof.

- At most f processors may fail
- But the number of rounds of the algorithm is $f + 1$
- This implies that there is at least one round r in which no processor fails
- From Lemma 1, we have that $W_i(r) = W_j(r)$, for all $u_i, u_j \in A(r)$
- From Lemma 2, we have that $W_i(f + 1) = W_j(f + 1)$ for all $u_i, u_j \in A(f + 1)$ □

FloodSet Correctness

- We now conclude correctness

Theorem. FloodSet solves the agreement problem for (at most f) crash failures.

Proof.

- Termination is obvious, because all processors explicitly decide in round $f + 1$
- For validity, if all start with the same value s, then all have $W = \{s\}$ initially and no other value can ever be sent in the system
  - Therefore, all active have $W = \{s\}$ in the end
  - and thus all decide s
- For agreement, let u_i and u_j be any two processors that decide
- This means that u_i and u_j are active after $f + 1$ rounds
- By Lemma 3, $W_i(f + 1) = W_j(f + 1)$ and both decide the same value (the unique value in the sets if singleton sets, otherwise s_0)

□

Summary

- Distributed problems may become substantially more difficult in the presence of failures
  - Sometimes even impossible to solve
  - Impossibility results are considered the “pearls” of Distributed Computing
- Main types:
  - Communication/link failures
    - e.g., lost messages
  - Processor failures
    - e.g., crash failures, Byzantine failures
- In the case of possibly unbounded lost messages
  - Agreement is impossible to solve
  - Coordinated attack problem
- In case of bounded crash failures
  - FloodSet solves agreement in complete networks
  - $f + 1$ rounds
  - $O((f + 1)n^2)$ messages