Lecture 9-10: Fault Bounds of Consensus

CS 539 / ECE 526

Distributed Algorithms
Today: Fault Bounds

• How many faults can we tolerate?
• Highly sensitive to various conditions
• All the fault bounds in this lecture are tight
Outline

• Fault bounds in synchrony
  – Byzantine agreement
  – Byzantine without signatures
  – Total-order broadcast and Replication

• Fault bounds in asynchrony
  – Broadcast
  – All other problems (FLP impossibility)

• Partial synchrony
  – Crash
  – Byzantine
Fault Bounds So Far

• Synchronous crash broadcast: $f < n$ (flooding)

• Synchronous Byzantine broadcast with signatures: $f < n$ [Dolev-Strong, 1983]

• How about Agreement?

• How about without signature?

• How about asynchrony?
Recall Agreement

• n parties, each has an input $x_i$, up to f faulty

• Safety: no different outputs

• Liveness: everyone outputs

• Validity: every honest inputs $x \rightarrow$ every honest outputs $x$
Recall Agreement Validity

• Every honest inputs \( x \) \( \rightarrow \) every honest outputs \( x \)

• Some examples: what should the output be given following inputs?
  – Binary inputs: 1, 1, 1, 1, 1?
    • Must be 1
  – Binary inputs: 0, 1, 1, 0, 1?
    • Must be 1 if both 0s are Byzantine inputs
    • Otherwise, either 0 or 1
  – Multi-value inputs: 3, 3, 5, 2, 3, 3, 3?
    • Must be 3 if 5 and 2 are Byzantine inputs
    • Otherwise, anything is fine
Recall Agreement Validity

• Every honest inputs $x \rightarrow$ every honest outputs $x$
  – Not meant to be useful
  – Just an easy condition to rule out trivial solutions

• Why don’t we define a more useful validity?

• Turns out it may make the problem too hard
  (problem set 2)
Broadcast to Agreement

- Byzantine broadcast (BB) gives BA if $f < n/2$
  - Every party invokes BB on its input
  - Every party gets an agreed upon vector
    - Byzantine $\rightarrow$ Any value in that position of vector
  - Everyone picks the most frequent value
    - $f < n/2$ needed for validity of Byzantine agreement
Broadcast to Agreement

• Safety: same vector, same way to pick

• Liveness: obvious

• Validity: if all honest have same input $x$, then $x$ will be the most frequent (since $f < n/2$)

• Round complexity: same as BB

• Communication complexity: $n$ times BB
Byzantine Agreement Fault Bound

• Byzantine agreement is not solvable if $f \geq n/2$
  – Proof: Divide parties into two groups $P$ and $Q$ such that $|P| \leq f$ and $|Q| \leq f$
  – Scenario I: $P$ are honest and receive input $v$; $Q$ are Byzantine and behave as if they receive input $v'$
  • $P$ commits $v$ due to validity
Byzantine Agreement Fault Bound

• Byzantine agreement is not solvable if \( f \geq n/2 \)

  – Proof: Two groups \(|P| \leq f\) and \(|Q| \leq f\)

  – Scenario I: P honest & receive \( v \), Q Byzantine & receive \( v' \) → P commit \( v \) due to validity

  – Scenario II: Q honest & receive \( v' \), P Byzantine & receive \( v \) → Q commit \( v' \) due to validity

  – Scenario III: P receive \( v \), Q receive \( v' \), both honest
    • P cannot distinguish III from I & commit \( v \)
    • Q cannot distinguish III from II & commit \( v' \)
Broadcast to Agreement

• Crash tolerant agreement for $f < n$ with a modification to validity
  – Every party invokes broadcast on its input
  – Every party gets an agreed upon vector
    • Crash $\rightarrow$ possibly $\perp$ in that position of vector
  – Everyone picks the most frequent non-$\perp$ value
Broadcast to Agreement

• Problem with standard validity when $f \geq n/2$
  – Example: inputs 0, 0, 0, 1, 1, 1. How to pick in a tie?
  – Pick 0? What if all three parties with input 0 crash right before they output?
    • All three non-faulty have input 1, must output 1
    • Symmetric problem for picking 1

• Modified validity: if all n parties input x, all non-faulty parties output x
Broadcast to Agreement

- Safety: same vector, same way to pick
- Liveness: obvious
- Validity: all $n$ parties input $x \rightarrow$ agreed-upon vector has only $x$ and $\perp \rightarrow$ all pick $x$ (non-$\perp$)

- Round complexity: same as broadcast
- Communication complexity: $n$ times broadcast
Fault Bound without Signatures

• BA or BB without signatures: \( f < \frac{n}{3} \)

[Lamport-Shostak-Pease, 1982]
Fault Bound without Signatures

• BA or BB without signatures: \( f < \frac{n}{3} \)

• Previous argument was handwavy
  – We are trying to prove \textit{No} algorithm works
  – Cannot assume how the protocol works

• Rigorous proof next [Fischer-Lynch-Merritt, 1986]
  – Step 1: no BA solution for \( n = 3, f = 1 \)
  – Step 2: generalize to any \( n \leq 3f \)
Fischer-Lynch-Merritt Proof

• Suppose for contradiction that there exists an algorithm that solves BA with $n = 3, f = 1$
Fischer-Lynch-Merritt Proof

• Connect six non-faulty processes in a ring, let them run the algorithm, and feed them inputs as in the figure

```
A 1 0 0 1
B 1 1 1 1
C 0 0 0 0
```
Fischer-Lynch-Merritt Proof
Fischer-Lynch-Merritt Proof
Fischer-Lynch-Merritt Proof

or 0?
Fischer-Lynch-Merritt Proof

• No algorithm solves BA with $n = 3$, $f = 1$
• Now generalize to any $n \leq 3f$
• Suppose for contradiction that a magic algo solves BA for some $n$ and $f$ where $n \leq 3f$
• We can use it to solve $1$-fault-out-of-$3$ BA
Fischer-Lynch-Merritt Proof

• Use f-out-of-n BA algo to solve 1-out-of-3 BA
  – Each of the three parties simulates $\leq f$ parties so that the total number of parties is n
  • 1 fault out of 3 $\rightarrow \leq f$ faults out of n
  – Run magic algo, 1-fault-out-of-3 BA solved
  – Contradiction, QED

• Where does the proof break down if using signatures?
Fault Bounds So Far

• Crash broadcast and agreement: $f < n$
• Byzantine broadcast (BB) with signatures: $f < n$
• Byzantine agreement (BA): $f < n/2$
• BA or BB without signatures: $f < n/3$

• Now moving on to more practical problems
Broadcast to Replication

• Broadcast gives replication

• Idea: Parties take turns to broadcast values
  – Crashed broadcaster $\rightarrow$ possibly $\perp$ in that position
  – Byzantine broadcaster $\rightarrow$ possibly invalid value
  – Everyone agrees on those, can simply discard

• This achieves Total-Order Broadcast
Total-Order (Atomic) Broadcast

• Parties propose values, and agree on a sequence of values

• Safety: no different values at every position in the sequence

• Liveness: every proposed value eventually added to the sequence

• Validity not needed (no trivial solution)
TO Broadcast vs. Replication

• TO broadcast: parties propose values, and agree on a sequence of values
  – Very close to replication, one subtlety remains

• Replication needs to serve external clients, not just reach consensus among servers
  – Clients do not see inner-working of the protocol
Replication

- **External clients** propose values (to servers) and **external clients** agree on a sequence of values.
Replication

- **External clients** propose values (to servers) and **external clients** agree on a sequence of values

- Safety: no different values at every position in the sequence

- Liveness: every proposed value eventually added to the sequence

- Validity: external (application level)
Replication

• Clients send values to servers; servers run a total-order broadcast and reply to clients
  – Problem solved for crash faults
  – Byzantine server can send a fake reply
    • Solution: require same reply from f+1 servers
Replication Fault Bound

• Byzantine fault tolerant replication requires same reply from \( f+1 \) replicas

• Need \( n > 2f \) so that honest > Byzantine

• Byzantine replication impossible if \( f \geq n/2 \)
  
  – Two groups \( |P| \leq f \) and \( |Q| \leq f \) present different views
  
  – Client don’t know who to believe

• Cannot distinguish the \( f \) Byzantine servers from the (up to) \( f \) honest servers
Fault Bounds for Synchrony

- Crash: $f < n$ (ignore agreement)
- Byzantine without signatures: $f < n/3$
- Byzantine with signatures:
  - Broadcast and total-order broadcast: $f < n$
  - Agreement and replication: $f < n/2$
- Moving on to asynchrony
Recall Asynchrony

• Any message can take arbitrarily long
  – but will eventually arrive
  – (Asynchrony also says any local computation can be arbitrarily long. But can be lumped into msg delay.)

• Helpful to think of asynchrony as an adversarial network scheduler
Broadcast in Asynchrony

• Cannot tolerate a single crash (broadcaster)
  – Same proof as in async impossibility of synchronizer
  – No msg from broadcaster, what do we do?
FLP Impossibility

- Under asynchrony, no deterministic agreement protocol can tolerate a single crash fault [Fischer-Lynch-Patterson, 1985]

- Recall configuration and valency
  - Step 1: there exists an initial bivalent config
  - Step 2: can always stay bivalent
Recall Configurations

• Union of the states of all parties

• A protocol execution is an evolution of configurations: $C_0 \rightarrow C_1 \rightarrow C_2 \ldots$

• In synchrony, evolve after each round

• In asynchrony, evolve after each msg arrival
  – “Msg m arrives at party p” is called an “event”
More on Async Configurations

• \( C_0 \rightarrow_e C_1 \rightarrow_e' C_2 \)

• Apply events in what order? Does it matter?

• Must apply \( e \) before \( e' \) if \( e \) happens before \( e' \)
  – Type 1: two events with the same recipients
  – Type 2: one event “triggers” another

• Otherwise, apply in either order, same outcome
  – \( C \rightarrow_e C_1 \rightarrow_e' C_2 \) \hspace{1cm} \( C \rightarrow_{e'} C'_1 \rightarrow_e C_2 \)
Recall Valency

• A config C is 0-valent, if in all configs reachable from C, honest parties decide 0
  – No matter what happens from now on, decide 0

• A config C is 1-valent, if ......, all decide 1

• Univalent = 0-valent or 1-valent

• Bivalent = not univalent
FLP Impossibility Proof

• Step 1: there exists an initial bivalent config
  – Proved in round lower bound

• Step 2: can always stay bivalent
  – What do we have to prove exactly?
  – ∀ bivalent C, ∃ bivalent C’ such that C → C’ ?
A Warm-Up (Not Actual Proof)

• ∀ bivalent C, ∃ bivalent C’ such that C → C’
  – Suppose for contradiction all evolution of C univalent
  – ∃ e₀, e₁ s.t. C → e₀ C₀ (0-val) and C → e₁ C₁ (1-val)
  – If e₀ || e₁, then C → e₀ C₀ → e₁ C* == C → e₁ C₁ → e₀ C*
    • C* cannot be both 0-val and 1-val, contradiction
  – e₀ and e₁ could not have triggered one another if they both already exist (applicable to C)
  – e₀ and e₁ must have the same recipient p
A Warm-Up (Not Actual Proof)

• ∀ bivalent C, ∃ bivalent C’ such that C → C’

  – Suppose for contradiction all evolution of C univalent

  – ∃e₀, e₁ with the same recipient p such that
    
    C → e₀ C₀ (0-val) and C → e₁ C₁ (1-val)

  – Fate of system depends on which msg reaches p first

    • Must wait for p to tell us. What if p does not speak?
    
    • Can’t wait forever; Any decision could be wrong

  – Contradiction. C must have a bivalent evolution
FLP Impossibility Proof

• Step 1: there exists an initial bivalent config

• Step 2: can always stay bivalent
  – What do we have to prove exactly?
  – $\forall$ bivalent $C$, $\exists$ bivalent $C'$ such that $C \rightarrow C'$ ?
    • Insufficient: may be delaying some events forever

• Actual Step 2: $\forall$ bivalent $C$, $\forall e$ applicable to $C$, $\exists$ bivalent $C'$ such that $C \rightarrow \ldots \rightarrow_e C'$ !
  – All msgs eventually delivered, still bivalent!
FLP Impossibility Proof

• ∀ bivalent C, ∀ e applicable to C, ∃ bivalent C’ such that C \rightarrow ... \rightarrow_e C’
  – S: set of configs reachable from C w/o applying e
  – T: set of configs by applying e to S
  – Want to prove T contains a bivalent config

• Proof:
  – Suppose for contradiction all configs in T univalent
  – Can find S_0 and S_1 \in S s.t. S_i \rightarrow_e is i-valent
    • Find 0-val A_0 reachable from C. If A_0 \in S, done;
      Else, trace back to the config before applying e
FLP Impossibility Proof

• \( \forall \) bivalent \( C \), \( \forall e \) applicable to \( C \), \( \exists \) bivalent \( C' \) such that \( C \rightarrow \ldots \rightarrow_e C' \)
  – \( S \): set of configs reachable from \( C \) w/o applying \( e \)
  – \( T \): set of configs by applying \( e \) to \( S \)
  – Suppose for contradiction all configs in \( T \) univalent
  – Can find \( S_0 \) and \( S_1 \in S \) s.t. \( S_i \rightarrow_e \) is i-valent
  – Can find neighboring \( S_0' \) and \( S_1' \in S \) s.t. \( S_0' \rightarrow_e \), \( S_1' \) and \( S_i' \rightarrow_e \) is i-valent
    • \( S \) is connected, such neighbors must exist
  – \( S_0' \rightarrow_e \) is 0-valent, \( S_0' \rightarrow_e \), \( S_1' \rightarrow_e \) is 1-valent
  – Rest of the proof same as warm-up
FLP Impossibility Proof

– $S_0' \rightarrow_e$ is 0-valent, $S_0' \rightarrow_{e'}$, $S_1' \rightarrow_e S^*$ is 1-valent

Rest same as warm-up:

– $e \# e'$, otherwise $S^*$ is both 0-val and 1-val

– So $e$ and $e'$ have the same recipient $p$

– Fate depends on which msg arrives at $p$ first

– What if we don’t hear from $p$?

– Can’t tell if $p$ crashed or is just slow

– Can’t wait forever; Any decision could be wrong
FLP Impossibility

• FLP does not say asynchronous consensus is impossible! Randomized consensus possible.

• Where does the proof rely on “deterministic”?

• Does it mean every deterministic protocol ALWAYS fails under asynchrony?
  – No, just says it can fail, can also get lucky.
What can we do given FLP?

• Consider easier problems

• Randomization
  – asynchronous agreement, total order broadcast, and replication possible under randomization
  – Single-value broadcast still impossible

• Consider easier models (partial synchrony)
  – Single-value broadcast still impossible under psync
Partial Synchrony

• (Intuitively) The network is sometimes asynchronous and sometimes synchronous
  – Maintain safety during asynchronous periods
  – Achieve liveness during synchronous periods
Partial Synchrony

• (Formally) There exists an unknown Global Standardization Time (GST) after which the network becomes synchronous
  – Forever synchronous after GST???
    • Hope to capture “sufficiently long sync periods”
  – Unknown to whom?
    • Can be viewed as a game between protocol designer and the adversary
Psync Agreement Fault Bound

• Crash: $f < \frac{n}{2}$
  
  – Proof: Two groups $|P| \leq f$ and $|Q| \leq f$
  
  – Scenario I:
  
  – Scenario II:
  
  – Scenario III:
Psycopg Agreement Fault Bound

• Crash: $f < \frac{n}{2}$
  – Proof: Two groups $|P| \leq f$ and $|Q| \leq f$
  – Scenario I: $P$ non-faulty & receive $v$, $Q$ crash
    • $P$ eventually commit $v$ due to validity
  – Scenario II: $Q$ non-faulty & receive $v'$, $P$ crash
    • $Q$ eventually commit $v'$ due to validity
  – Scenario III: Both non-faulty, $P$ receive $v$, $Q$ receive $v'$
    GST sufficiently large $\Rightarrow$ Both think the other crashed
    • $P$ commit $v$, $Q$ commit $v'$
Psync Agreement Fault Bound

• Byzantine: $f < n/3$
  – Proof: Three groups $|P| \leq f$, $|Q| \leq f$, $|R| \leq f$
  – Scenario I:
    – Scenario II:
    – Scenario III:
Psync Agreement Fault Bound

• Byzantine: \( f < \frac{n}{3} \)
  – Proof: Three groups \(|P| \leq f, |Q| \leq f, |R| \leq f\)
  – Scenario I: P/R non-faulty & receive \( v \), Q crash
    • P eventually commit \( v \) due to validity
  – Scenario II: Q/R non-faulty & receive \( v' \), P crash
    • Q eventually commit \( v' \) due to validity
  – Scenario III: P non-faulty & receive \( v \), Q non-faulty & receive \( v' \), R Byzantine behave towards P like in I and towards Q like in II. GST sufficiently large.
    • P cannot distinguish from I, commit \( v \)
    • Q cannot distinguish from II, commit \( v' \)
Async and Psync Fault Bounds

• Agreement under partial synchrony
  – Crash: $f < \frac{n}{2}$
  – Byzantine: $f < \frac{n}{3}$ (nothing to do with signatures)

• Both bounds apply to async or randomized

• Both bounds apply to TO-bcast and replication
  – Standard (single-value) broadcast still cannot tolerant even a single crash!
Fault Bounds Summary

• Async deterministic: $f = 0$
  – Broadcast, agreement, total-order bcast, replication

• Psync or randomized async
  – Broadcast: $f = 0$
  – Agreement, total-order broadcast, or replication: crash: $f < n/2$, Byzantine: $f < n/3$

• Sync
  – Crash: $f < n$ for all four problems
  – Byzantine no signature: $f < n/3$ for all four problems
  – Byzantine with signature
    • $f < n$ for broadcast and total-order broadcast
    • $f < n/2$ for agreement and replication
Fault Bounds Better Summary

- Byzantine agreement: $f < n/2$
- Byzantine replication: $f < n/2$
- Byzantine no signature: $f < n/3$
- Async deterministic: $f = 0$
- Psync broadcast: $f = 0$
- Psync crash: $f < n/2$
- Psync Byzantine: $f < n/3$