ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Lecture 6: State Machine Replication and Consensus

CS 539 / ECE 526

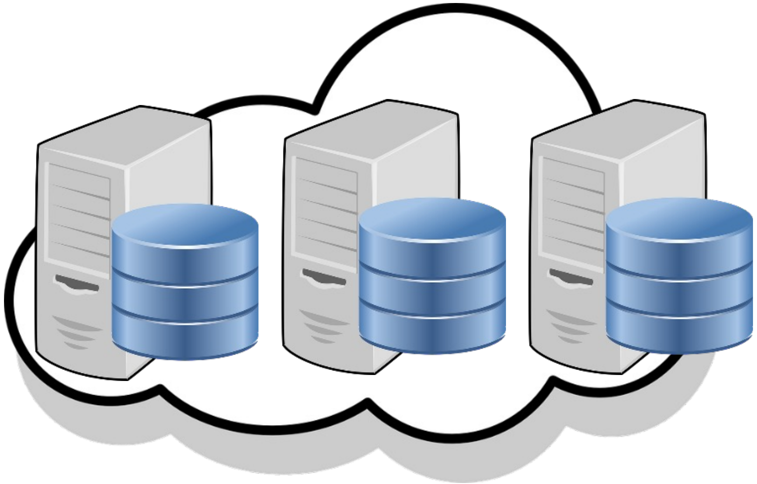
Distributed Algorithms

Outline

- Motivation and Model
- Difficulty with Link Failure
- Byzantine agreement and broadcast

(State Machine) Replication

- Consider any service
  - The server may fail
- Replicate the service
  - Need consensus
  - Despite some faulty servers
- Goal: provides an illusion of a single non-faulty server despite that some servers are faulty



(State Machine) Replication

- Goal: provides an illusion of a single non-faulty server despite that some servers are faulty
- More formally: all servers commit the same sequence of "values"
  - Will start with a simpler variant: agree on a single value

Types of Process Faults

- Crash: at some point the process stops executing
  - Msgs need to sent one at a time, so may stop after sending a subset of msgs in last (lockstep) round
  - But need not worry about stopping in the middle of sending a msg
    - Invalid msgs can be detected and discarded

Types of Process Faults

- Crash: at some point the process stops executing
- Byzantine: arbitrary behavior, malicious
  - Hardest type of fault to deal with

Types of Process Faults

- Crash: at some point the process stops executing
- Byzantine: arbitrary behavior, malicious
- Other faults (that we will not focus on)
  - Fail-stop: notify other processes before crashing
  - Crash-recovery
  - Omission

“Right” Model for Replication?

- Traditionally:
  - Message passing
  - Asynchrony (or close to it)
  - Crash faults
  - Generic graph for theoretical interests, complete graph also reasonable with crash and async
  - Known set of participants
  - Reliable links

Some History

- Consensus problem introduced before 1980
- Lots of interests/progress in 1980s and 1990s
- Reduced interests in 2000s
  - Crash fault tolerance replication mostly solved (and sees wide adoption later)
  - Byzantine fault tolerance (BFT) no justification application
- … Until Nakamoto's Bitcoin (2009) revived BFT with new applications: decentralized X/Y/Z …
  - Bitcoin assumes some degree of synchrony
  - Set of participants unknown or even changing

“Right” Model for Replication?

- Traditionally:
  - Message passing, asynchrony (or close to it), crash faults, generic or complete graph, reliable links, fixed and known participants
- More recently:
  - Synchrony, asynchrony, and more
  - Crash faults, Byzantine faults, and more
  - Unknown and changing participants

Timing Model

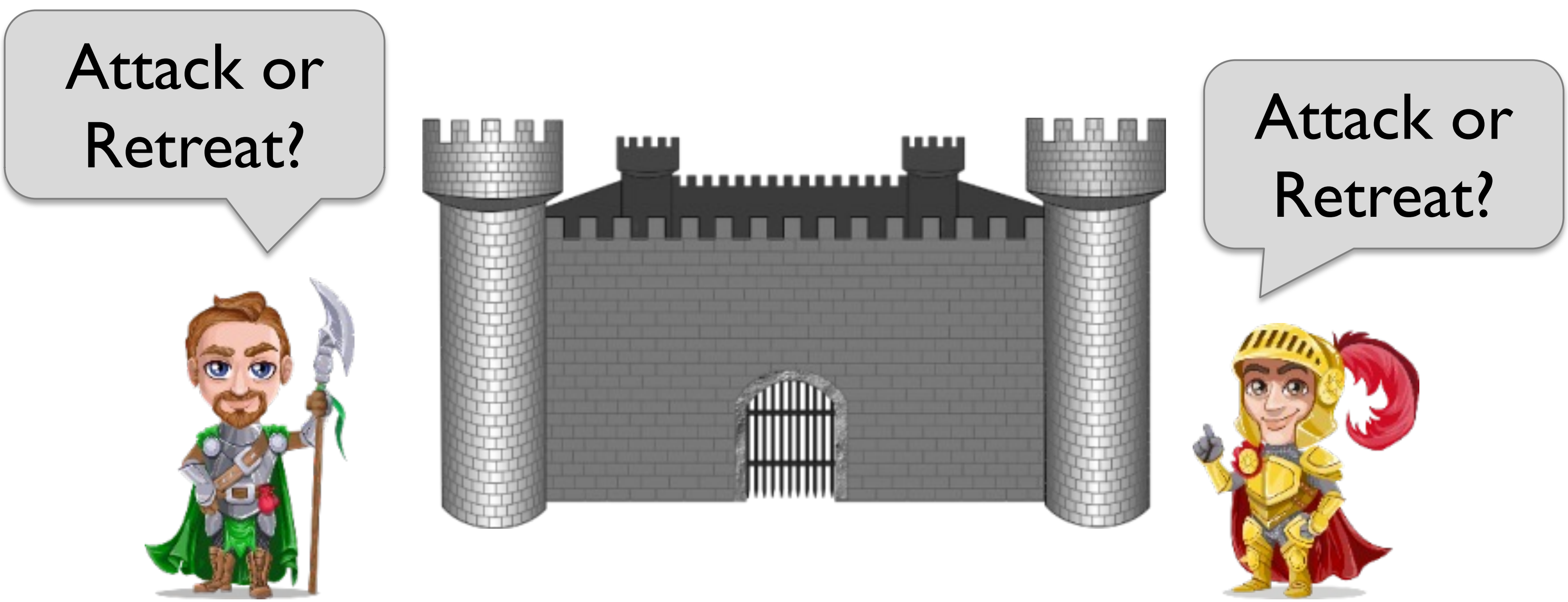
- Sufficient to focus on communication delay
  - Lump computation delay into communication delay
- Synchrony: delay upper bound Δ for every msg known to all parties
  - More ideal model: lockstep rounds
- Asynchrony: no upper bound on delay
  - Every message can take arbitrarily long but eventually arrives (reliable links)
- Partial synchrony: alternating periods of synchrony and asynchrony

Outline

- Motivation and Model
- Difficulty with Link Failure
- Byzantine agreement and broadcast

Two General Agreement Problem

- Two generals coordinate an attack
  - Both generals are honest
  - Messenger may be captured



Two General Agreement Problem

- Two honest generals each has an input
- The link between them may lose messages
- Desired outcome: two generals same output
- Safety: the two generals do not output different values
- Liveness: every general outputs a value
- Validity: If the two generals both input x, then they both output x
  - Needed to avoid trivial solutions

Two General Impossibility

- Surprisingly, not solvable deterministically

- Theorem: No deterministic algorithm can solve the two general problem with a lossy link
  - Even with lockstep synchrony and one-bit inputs

- In general, making the problem easier makes an impossibility result stronger

Two General Impossibility Proof

- Suppose for contradiction such an algo exists
  - WLOG, can assume each general sends a msg every round (can send NoMsg)
- Consider its execution in which both generals input 1 and all msgs arrive
  - Both generals output 1 due to validity
  - Suppose this execution terminates after m rounds, call it E_{2m}

Two General Impossibility Proof

- Suppose for contradiction such an algo exists
- Consider its execution in which both generals input 1 and all msgs arrive (call it E_{2m})
- E_{2m-1}: last msg 1→2 lost (lossy link)
  - Indistinguishable from E_{2m} to General 1
  - General 1 outputs 1 (in round m, and terminates)
  - General 2 outputs 1 due to safety

Two General Impossibility Proof

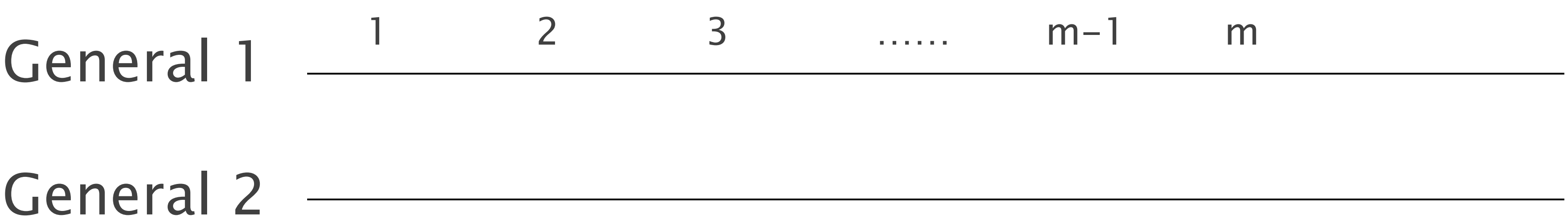
- Suppose for contradiction such an algo exists
- Consider its execution in which both generals input 1 and all msgs arrive (call it E_{2m})
- E_{2m-1}: last msg 1→2 lost (lossy link)
- E_{2m-2}: last msg 2→1 also lost (lossy link)
  - Indistinguishable from E_{2m-1} to General 2
  - General 2 outputs 1
  - General 1 outputs 1 due to safety

Two General Impossibility Proof

- Suppose for contradiction such an algo exists
- Consider its execution in which both generals input 1 and all msgs arrive (call it E_{2m})
- E_{2m-1}: last msg 1→2 lost (lossy link)
- E_{2m-2}: last msg 2→1 also lost (lossy link)
- Remove msg one by one, each time one general cannot distinguish from previous exec

Two General Impossibility Proof

- Suppose for contradiction such an algo exists
- Consider its execution in which both generals input 1 and all msgs arrive (call it E_{2m})
- Remove msg one by one, each time one general cannot distinguish from previous exec
- E_0: both input 1, all msgs lost, both output 1
- E': general 2 inputs 0, all msgs lost



Two General Impossibility Proof

- Suppose for contradiction such an algo exists
- Consider its execution in which both generals input 1 and all msgs arrive (call it E_{2m})
- Remove msg one by one, each time one general cannot distinguish from previous exec
- E_0: both input 1, all msgs lost, both output 1
- E': general 2 inputs 0, all msgs lost
  - General 1 cannot distinguish from E_0, still outputs 1
  - General 2 has to output 1; otherwise safety violated

Two General Impossibility Proof

- Suppose for contradiction such an algo exists
- Consider its execution in which both generals input 1 and all msgs arrive (call it E_{2m})
- Remove msg one by one, each time one general cannot distinguish from previous exec
- E_0: both input 1, all msgs lost, both output 1
- E': general 2 inputs 0, all msgs lost, outputs 1
- E'': general 1 also inputs 0, all msgs lost
  - General 2 cannot distinguish from E', still outputs 1!
  - Validity violated! Contradiction. QED

Two General Impossibility

- Theorem: No deterministic algorithm can solve the two general problem with a lossy link
  - Even with lockstep synchrony and one-bit inputs
  - Where did the proof rely on deterministic?

- Randomization helps a little, not by much (will not go into this)

- Became a justification for reliable links
  - Lossy links too hard to solve?

Justification for Reliable Links

- But … this is not sound reasoning
- When generalized to *n* honest generals, impossibility holds only if ALL links are lossy
- Fraction of lossy links overlooked, more research is needed

Justification for Reliable Links

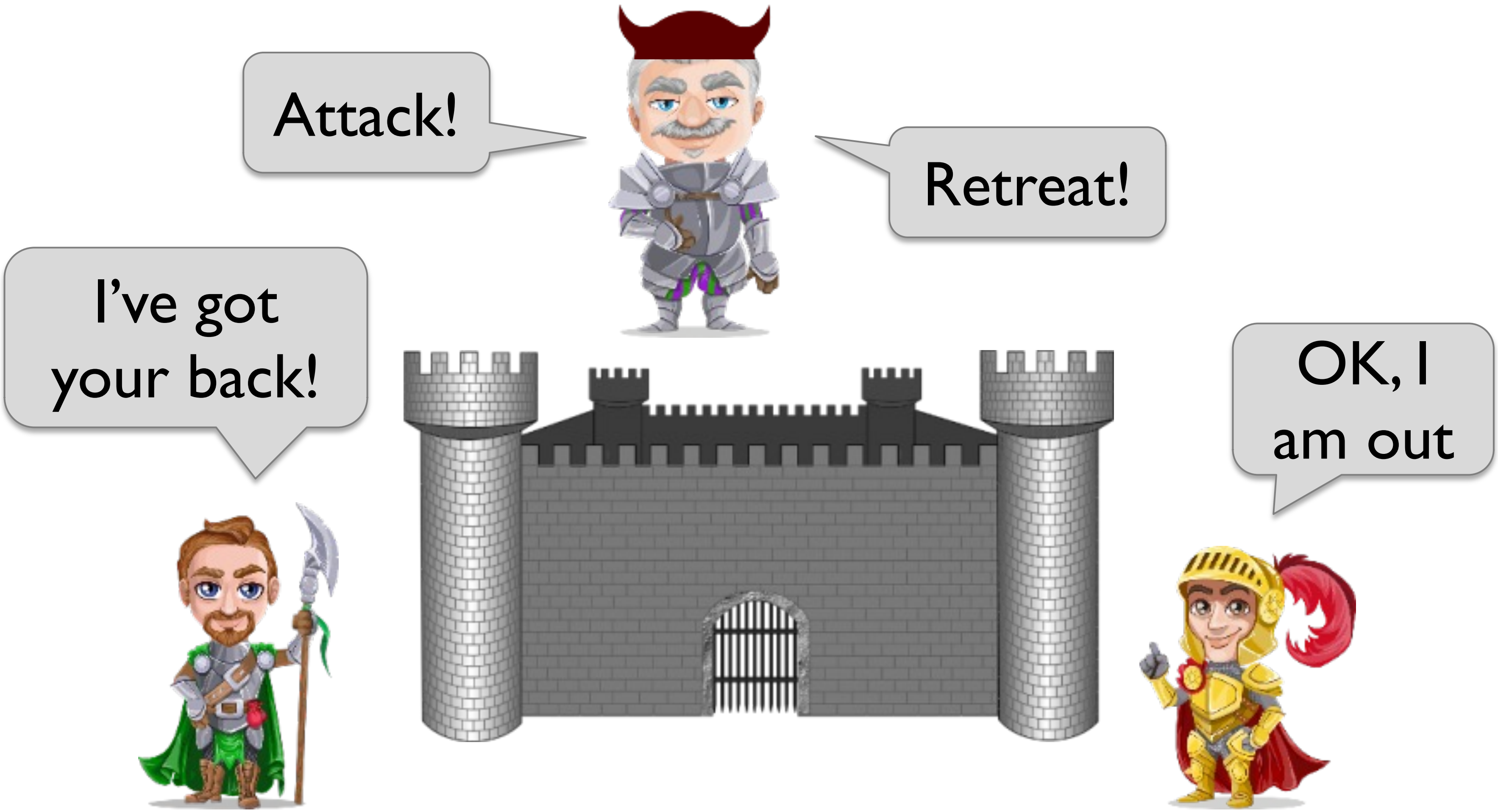
- There is, however, a reasonable justification for assuming reliable links
- A process can keep re-sending until receiving an ack from recipient
- Turns a lossy link into a reliable async link! (From a practical perspective)

Outline

- Motivation and Model
- Difficulty with Link Failure
- Byzantine agreement and broadcast

Byzantine General's Problem

- [Lamport, Shostak, and Pease 1982]



Byzantine Agreement Problem

- n generals, each has an input value
- Up to f of them can be traitors

- Desired outcome: every **honest** general outputs the same value

Byzantine Agreement Problem

- n generals, each has an input value
- Up to f of them can be traitors

- Safety: no two honest generals output different values
- Liveness: every honest general outputs a value
- Validity: if every honest general inputs x, then every honest general outputs x
  - Needed to avoid trivial solutions

Byzantine Agreement Problem

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

- Safety: no different outputs
- Liveness: everyone outputs
- Validity: every honest inputs x → everyone outputs x

Byzantine Broadcast Problem

- n generals, including a commander
- Commander has an input value x
- Up to f of them (including the commander) can be traitors

- Safety: no two honest generals output different values
- Liveness: every honest general outputs a value
- Validity: if the commander is honest, every honest general outputs x

Byzantine Broadcast Problem

- n parties, including a designated sender with an input x, up to f faulty

- Safety: no different outputs
- Liveness: everyone outputs
- Validity: sender honest → everyone outputs x

Remarks

- Early papers are inconsistent in terminology! Check their actual definitions!

- Usually assume parties know n and f
- But parties do not know *who* are faulty
  - Otherwise problem is trivial
- Can a Byzantine party behave honestly?
  - Yes, by definition
- Is it still considered Byzantine?
  - Yes. There is no requirement on what they output.

Remarks on Validity

- Broadcast validity seems natural and useful
  - Sender honest → output sender's value

- Agreement validity … much less clear
  - Every honest inputs x → every honest outputs x
  - Is this useful?
  - Let's look at some examples first. What should the output be given following **honest** inputs?
    - Binary inputs: 1, 1, 1, 1, 1?
    - Binary inputs: 0, 1, 1, 0, 1?
    - Multi-value inputs: 3, 3, 5, 2, 3, 3, 3?

Remarks on Validity

- Broadcast validity seems natural and useful
  - Sender honest → output sender's value

- Agreement validity … much less clear
  - Every honest inputs x → every honest outputs x
  - Is this useful?
  - Let's look at some examples first. What should the output be given following **honest** inputs?
    - Binary inputs: 1, 1, 1, 1, 1? Must be 1
    - Binary inputs: 0, 1, 1, 0, 1? Either 0 or 1 is OK
    - Multi-value inputs: 3, 3, 5, 2, 3, 3, 3? Anything!

Remarks on Validity and Usefulness

- Broadcast validity seems natural and useful
- Agreement validity … not really, only useful in very limited situations

- Meant to be a clean and easy problem
  - Easiest validity to forbid trivial solution
  - Value lies in the techniques, usually shed light on solving replication
  - Also valuable in impossibility proofs

Tolerating Faults is Hard!

- In general, when there are faults, we almost always study the consensus problem. Why?
- Partly because it is the easiest problem!
- But still quite hard! (and deceptively simple)

- Let us start from the simplest model
  - f crash faults out of n parties in total
  - Pair-wise reliable links, lockstep synchrony
  - Binary input: x is 0 or 1
- Try to come up with an algorithm!