Proposer

Send `IAmLeader(n)` to all
Wait for a majority of responses

Accept

If $n$ is the highest leader # I have seen: respond with `YouAreLeader(Value, LeaderWhoProposedValue)`

Once majority is received, send `Propose(n, V)` where $V$ is the highest-leader proposal among the responses (or my own value, if none of the responses had a value)

If $n$ is the highest leader # I have seen, send `Accept(n, V)` to the learner
The original Paxos algorithm achieves agreement on one value.

SMR required replicas to agree on the sequence of commands that will be executed.

3. Ensure that all replicas go through the same sequence of state transitions.

MultiPaxos: Run an instance of Paxos for each slot in the sequence.

**Important**: we don’t need to run phase 1 (election) every time!
Proposers, acceptors and learners are all collocated on $2f + 1$ replicas
Paxos/SMR in Real Life

Proposers, acceptors and learners are all collocated on $2f + 1$ replicas.
BYZANTINE FAULT TOLERANCE
A HIERARCHY OF FAILURE MODELS

- Fail-stop
- Crash
- Send omission
- Receive omission
- General omission
- Arbitrary (Byzantine) failures

○ = benign failures
WHAT ARE BYZANTINE FAILURES

The short answer: they can be anything!
(they can even be crash/omission failures)

Examples of commission failures

- A bit flip in memory
- Manufacturing defect
- Alpha particles
- Network card malfunction
- Intentional behavior
  - Rational node: trying to game the system for personal gain
  - Malicious node: trying to bring the system down
THE BYZANTINE GENERALS

- Synchronous communication
- One general may be a traitor
The Byzantine Generals

- Synchronous communication
- One general may be a traitor
- One of the generals is the commander C
  - The commander decides **Attack** or **Retreat**

Goals

1. If C is trustworthy, every trustworthy general must follow C’s orders
2. Every trustworthy general must follow the same battle plan
Remember when things were simpler?
You can’t trust anyone these days...
YOU CAN’T TRUST ANYONE THESE DAYS…

C

G1

He said “retreat”

He said “attack”

G2

Attack

Retreat
YOU CAN’T TRUST ANYONE THESE DAYS…

He said “retreat”
He said “attack”
Attack
Retreat

C

G₁

G₂

C

G₁

G₂

Attack

He said “retreat”
“But they were all of them deceived…”
A LOWER BOUND

Theorem
There is no algorithm that solves TRB for Byzantine failures if \( n \leq 3f \)

Lamport, Shostak and Pease, The Byzantine Generals Problem, 1982
Problem set #1 graded

- Regrade requests: written explanation, hand in to me by Wednesday 10/16
- We will regrade the entire problem set (your grade may go up or down as a result)

Upcoming deadlines

- Declare project topic: tomorrow
- Problem set #2: 10/16
- Midterm exam: 10/21
- Implementation project: 10/30
- Presentation slides: 11/2
Administrivia

Presentations start on 10/23

- Schedule posted on Piazza
- 2 papers in each class
- Write a short review for one of them
  - You don’t have to write a review for the lecture during which you are presenting
- We will setup a website for you to post reviews
WRITING REVIEWS

- Short summary
- Strengths
  - what is the contribution
  - nice insight/implementation/presentation
- Weaknesses
  - I think this doesn’t work, because…
  - performance is bad
  - indifferent motivation
  - bad presentation
- (optional) future work?
PBFT: A Byzantine Renaissance

Practical Byzantine Fault Tolerance
(Castro, Liskov 1999-2000)

• First practical protocol for asynchronous BFT replication

• Like Paxos, PBFT is safe all the time, and live during periods of synchrony
Barbara Liskov
Turing Award 2008
THE SETUP

**System model**
- Asynchronous system
- Unreliable channels

**Crypto**
- Public/private key pairs
- Signatures
- Collision-resistant hashes

**Service**
- Byzantine clients
- Up to $f$ Byzantine servers
- $n = 3f + 1$ total servers

**System goals**
- Always safe
- Live during periods of synchrony
The general idea

- One primary, 3f replicas
- Execution proceeds as a sequence of **views**
  - A view is a configuration with a well-defined primary
- Client sends signed commands to primary of current view
- Primary assigns sequence number to client’s command
- Primary is responsible for the command eventually being decided
What could possibly go wrong!?

• The primary could be faulty!
  ➤ could ignore commands, assign same sequence number to different requests, skip sequence numbers, etc.
  ✔ Backups monitor primary’s behavior and trigger view changes to replace a faulty primary

• Replicas could be faulty!
  ➤ could incorrectly forward commands received by a correct primary
  ✔ any single request may be misleading; need to rely on quorums of requests
  ➤ could send incorrect responses to the client
  ✔ client waits for $f + 1$ matching responses before accepting