• **CS244b slack workspace** is up

• My signup sheet done, please feel free to meet with me by appointment

• Please sign up to meet with Jim in a couple of weeks

• Please ask lots of questions today!
  - Jim please interrupt if I miss raised hands
  - Should be a whiteboard lecture, but issues with tablet/handwriting from last lecture
  - Not intended to go at “slide lecture” pace
  - But very weak feedback loop from zoom lectures
FLP: “pick \( \leq 2 \) of Safety, Liveness, Fault-tolerance\(^1\)”

So far have sacrificed liveness (Paxos, Raft, PBFT)
- Want safety, fault-tolerance always
- Settle for termination \textit{in practice} (and avoid stuck states)
- \textit{Partial} and \textit{weak} synchrony can help (e.g., PBFT)

Two more ideas:
- Remove asynchronous assumption entirely [Byzantine generals]
- Remove deterministic assumption

Learning goals for today
- Learn about randomized \textit{asynchronous} protocols (how they work, pros, cons)
- Give you lots of useful tools (threshold crypto, erasure coding, reliable broadcast, common coins, async. binary agreement, …)

\(^1\)in a \textit{deterministic, asynchronous} protocol
Byzantine generals problem [Lamport’82]

- Commander $G_0$ sends a message to lieutenants $\{G_1, \ldots, G_n\}$
  - Either all honest generals must attack, or all must retreat
  - Some generals could be faulty, including commander
  - But non-faulty nodes communicate in time $T$ by everyone’s clock
    (So $T - \epsilon$ real time to account for clock skew)

- First insight: w/o digital signatures, need more than 3 nodes
  - Else, $G_1$ and $G_2$ can’t prove to each other what commander said
Byzantine generals w. signatures

- **Warm-up exercise: 0 faulty generals**
  - $G_0$ broadcasts digitally signed order
  - Other nodes wait $T$ seconds, then follow order

- **If $\leq f$ faulty generals, go through $f + 1$ rounds $(0, \ldots, f)$:**
  - Round 0: $G_0$ broadcasts signed order $\langle v \rangle_{G_0}$
  - Round 1: Each other $G_i$ re-signs, broadcasts $\langle \langle v \rangle_{G_0} \rangle_{G_i}$
  - Round $r$: For each $m$ received in $r - 1$ with new value $v$
    ▶ $G_i$ ensures $m$ has $r + 1$ nested signatures of different nodes (or ignores)
    ▶ Adds own sign, broadcasts $\langle m \rangle_{G_i}$ ($r + 1$ nested sigs)
  - After round $f$, $G_i$ receives 0 or more valid messages
    ▶ Deterministically combine values and output result
      (e.g., take median or default to retreat if 0 valid messages)

- **$N$ nodes survives $f$ failures even if $N = f + 2$ (no 1/3 threshold)**
  - But loses safety if synchrony assumption is violated
  - That’s why most systems use partial/weak synchrony
Randomized protocols

- FLP proof considers delivering messages $m$ and $m'$ in either order
  - Assumes if different recipients, either order leads to same state
  - But logic only holds if messages are processed deterministically

- Paxos, Raft, PBFT “never get stuck”
  - Means there’s always some network schedule that leads to termination
  - So keep trying “rounds” (views, ballots, terms, etc.) until one terminates

- If network were random, we could talk about round termination probability
  - Unfortunately, network is hard to model / controlled by adversary
  - Can we instead make probability dependent on nodes’ random choices?
Asynchronous Binary Agreement (ABA)

- Simplest goal (agree on a single bit) still violates FLP
  - Ben Or first proposed sidestepping FLP with randomness…
- \(N\) nodes (\(\leq f\) faulty) each receive one bit input \(\{0, 1\}\)
  - Exchange messages and (ideally) output a bit
- Goals:
  - Agreement – if any non-faulty node outputs \(b\), all will
  - Termination – if all non-faulty nodes receive input, all output a bit
    - Since randomized, can terminate with probability \(1\)
    - E.g., infinite rounds each with finite termination probability
  - Validity – if all correct nodes received input \(b\), decision will be \(b\)
ABA surviving $f$ faults for $N > 5f$ nodes

Each node $i$ starts with input bit $x_i$, then executes:

```c
int x = x_i; // i’s input bit
for (round = 0;; ++round) {
    broadcast <VOTE, round, x>
    wait for $N-f$ VOTE messages in round (including i’s own)
    if more than $(N+f)/2$ VOTEs have same value $v$
        then broadcast <COMMIT, round, v>
    else broadcast <COMMIT, round, ?>
    wait for $N-f$ COMMIT messages in round (including i’s own)
    if more than $f+1$ COMMIT messages have same value $v \neq$ ?
        then set $x=v$; if more than $(N+f)/2$ COMMIT $v$
            then output $x$ as consensus value
        else set $x$ to a random bit // a.k.a. a coin flip
}
```

Why does this work?
Common coin [Rabin’83]

- Threshold crypto requires $N - f$ priv. key shares to sign/decrypt
  - Can encrypt/verify using only a single public key
  - Some deterministic/unique signatures algs work (e.g., RSA-FDH)
- Idea: Use threshold sig on $\langle \text{instance}, \text{round-number} \rangle$
  - Unpredictable but can be computed by any $N - f$ nodes
- Rabin’s trick: use common coin to randomize threshold
  - If bad network knows you need $(N + f)/2$ votes to decide, can ensure some nodes see over, some under threshold
  - But not if threshold random between $(N/2, N - 2f]$ (can repeat rounds to increase probability of success)
  - Base threshold on common coin computed after votes exchanged!
- Better algorithms include Mostéfaoui et al. (later)
- Caveat: setting up common coin requires trusted dealer
  - Or can use fancy crypto, but requires synchronous protocol
Reliable broadcast (RBC) [Bracha]

- Sender $P_S$ has input $h$ to broadcast to $N > 3f$ nodes \{${P_i}$\}
- Want: agreement, totality, validity [define these]
- Protocol
  1. $P_S$ broadcasts VAL($h$)
  2. $P_i$ receives VAL($h$), broadcast ECHO($h$)
  3. $P_i$ receives $N - f$ ECHO($h$) messages, broadcasts READY($h$)
  4. $P_i$ receives $f + 1$ READY($h$), broadcasts READY($h$) [if hasn’t already]
  5. $P_i$ receives $2f + 1$ READY($h$), delivers $h$
Reliable broadcast (RBC) [Bracha]

- **Sender** $P_S$ has input $h$ to broadcast to $N > 3f$ nodes $\{P_i\}$
- **Want:** agreement, totality, validity [define these]
- **Protocol**
  1. $P_S$ broadcasts $\text{VAL}(h)$
  2. $P_i$ receives $\text{VAL}(h)$, broadcast $\text{ECHO}(h)$
  3. $P_i$ receives $N - f$ $\text{ECHO}(h)$ messages, broadcasts $\text{READY}(h)$
  4. $P_i$ receives $f + 1$ $\text{READY}(h)$, broadcasts $\text{READY}(h)$ [if hasn’t already]
  5. $P_i$ receives $2f + 1$ $\text{READY}(h)$, delivers $h$
- **$N - f$ nodes includes majority of non-faulty nodes**
  - $\text{READY}$ from all non-faulty nodes has same $h$ $\implies$ agreement
  - If $P_S$ non-faulty, will all contain $P_S$’s input $h$ $\implies$ validity
- **If** $2f + 1$ nodes **send** $\text{READY}(h)$, **then** $f + 1$ **will be non-faulty**
  - Those $f + 1$ will make all non-faulty nodes to broadcast $\text{READY}(h)$
  - Since $N > 3f$, will get $2f + 1$ broadcasting $\text{READY}(h)$ $\implies$ totality
Why doesn’t RBC directly give us consensus?

- Each node RBCs its input; take median (like Byz. generals)

- Make $h$ a cryptographic hash

- Use Merkle tree so can verify each block of $h$

- Erasure coding: make $n > k$ shares of $k$-block msg, so any $k$ reconstruct msg [e.g., polynomial interpolation]

- Change protocol to send $\text{VAL}(h, b_i, s_i)$, broadcast $\text{ECHO}(h, b_i, s_i)$

- $s_i$ is share of message, $b_i$ is proof that it is in hash tree with root $h$

- Wait for $N - f$ $\text{ECHO}$ messages that permit reconstruction before sending $\text{READY}(h)$ (guaranteed a/f$_{\text{ter}}$/two.pnum f$_{\text{one.pnum}}$/one.pnum/zero.pnum / /one.pnum/four.pnum
Refining RBC

- Why doesn’t RBC directly give us consensus?
  - Each node RBCs its input; take median (like Byz. generals)
  - Don’t know when RBCs are done (else would violate FLP)
- What if $h$ is big and $P_S$ has to send many copies?
Refining RBC

• Why doesn’t RBC directly give us consensus?
  - Each node RBCs its input; take median (like Byz. generals)
  - Don’t know when RBCs are done (else would violate FLP)

• What if $h$ is big and $P_S$ has to send many copies?
  - Make $h$ a cryptographic hash
  - Use Merkle tree so can verify each block of $h$

• Erasure coding: make $n > k$ shares of $k$-block msg, so any $k$
  reconstruct msg [e.g., polynomial interpolation]
  - Change protocol to send $\text{VAL}(h, b_i, s_i)$, broadcast $\text{ECHO}(h, b_i, s_i)$
  - $s_i$ is share of message, $b_i$ is proof that it is in hash tree with root $h$
  - Wait for $N - f$ ECHO messages that permit reconstruction before
    sending $\text{READY}(h)$ (guaranteed after $2f + 1 \text{READY}(h)$)
let est = input_value // estimate of output value (0 or 1)
    r = 0 // round number (integer)
    RBC_results[] = infinite list of empty bit sets
thread_fork for(;;) {
    <EST, r', est'> <- RBC_receive
    add est’ to RBC_results[r]
}
for (int r = 0;; r++) {
    thread_fork RBC_broadcast <EST, r, est>
    wait until RBC_results[r] != {}, let w be in RBC_results[r]
    multicast <AUX, i, r, w>
    receive AUXes from N-f senders with w values in RBC_results[r]
    s <- common_coin(r) & 1 (low bit)
    if among N-f received AUXes have both w=0 and w=1
        est = s
    else if all have same value w {
        if w == s and haven’t output yet
            output(w) // but keep going
        est = w
    }
}
Asynchronous common subset (ACS)

- $N$ nodes $\{P_i\}$ get input, all output subset of inputs
  - Want: validity, agreement, totality

  while (fewer than $N-f$ RBCs have delivered a value
  && fewer than $N-f$ ABA instances have output 1) {
    if (RBC$_j$ delivers $v_j$)
      Supply 1 as input to ABA$_j$
  }
  Supply 0 as input to any remaining ABAs
  Output $\{ v_j \mid ABA_j \text{ output 1} \}$ [waiting for RBCs if needed]

- Why does this ACS work?

  $RBCs$ and $ABA$s output same at all non-faulty nodes $\implies$ agreement
  $N-f$ RBCs will deliver value (by totality of $RBC$s) $\implies$ totality
  $\therefore$ All nodes will exit the while loop
  $\therefore$ If $ABA_j = /one.pnum$ at any non-faulty node, then
  $RBC_j$ will deliver $v_j$
  $\geq N-f$ ABAs must output $/one.pnum$ $\implies$ validity
  $\therefore$ Hence at least $N-f/\text{two.pnum}$ must correspond to non-faulty nodes

$\text{one.pnum/\text{two.pnum} / /\text{one.pnum/four.pnum}}$
Asynchronous common subset (ACS)

- $N$ nodes $\{P_i\}$ get input, all output subset of inputs
  - Want: validity, agreement, totality

  while (fewer than $N-f$ RBCs have delivered a value
    && fewer than $N-f$ ABA instances have output 1) {
    if (RBC$_j$ delivers $v_j$)
      Supply 1 as input to ABA$_j$
  }

  Supply 0 as input to any remaining ABAs
  Output $\{ v_j | ABA_j \text{ output } 1 \}$ [waiting for RBCs if needed]

- Why does this ACS work?
  - RBCs and ABAs output same at all non-faulty nodes $\implies$ agreement
  - $N-f$ RBCs will deliver value (by totality of RBC) $\implies$ totality
    - All nodes will exit the while loop
    - If $ABA_j = 1$ at any non-faulty node, then $RBC_j$ will deliver $v_j$
  - At least $N-f$ ABAs must output 1 $\implies$ validity
    - Hence at least $N-2f$ must correspond to non-faulty nodes
Consensus from RBC and ACS

• **Strawman 1:**
  - Each $P_i$ uses RBC to broadcast $B$ oldest transactions
  - Use ACS to pick $N - f$ and take union of transactions
  - Problem?
• **Strawman 1:**
  - Each $P_i$ uses RBC to broadcast $B$ oldest transactions
  - Use ACS to pick $N - f$ and take union of transactions
  - Problem? Wastes lots of bandwidth sending $B$ around

• **Strawman 2:**
  - $P_i$ uses RBC on random $\lfloor B/N \rfloor$-sized subset of $B$ transactions
  - ACS as before
  - Problem?
**Strawman 1:**
- Each $P_i$ uses RBC to broadcast $B$ oldest transactions
- Use ACS to pick $N - f$ and take union of transactions
- Problem? Wastes lots of bandwidth sending $B$ around

**Strawman 2:**
- $P_i$ uses RBC on random $\lfloor B/N \rfloor$-sized subset of $B$ transactions
- ACS as before
- Problem? Network can censor victim transaction

**Solution?**
Consensus from RBC and ACS

- **Strawman 1:**
  - Each $P_i$ uses RBC to broadcast $B$ oldest transactions
  - Use ACS to pick $N - f$ and take union of transactions
  - Problem? Wastes lots of bandwidth sending $B$ around

- **Strawman 2:**
  - $P_i$ uses RBC on random $\lfloor B/N \rfloor$-sized subset of $B$ transactions
  - ACS as before
  - Problem? Network can censor victim transaction

- **Solution? Use threshold encryption**
  - Each node RBCs threshold encryption of $\lfloor B/N \rfloor$ transactions
  - Only decrypt after ACS complete
  - Threshold allows decryption even if sender fails
Algorithm HoneyBadgerBFT (for node $P_i$)

Let $B = \Omega(\lambda N^2 \log N)$ be the batch size parameter.

Let $PK$ be the public key received from TPKE.Setup (executed by a dealer), and let $SK_i$ be the secret key for $P_i$.

Let $buf := []$ be a FIFO queue of input transactions. Proceed in consecutive epochs numbered $r$:

// Step 1: Random selection and encryption

- let proposed be a random selection of $\lfloor B/N \rfloor$ transactions from the first $B$ elements of $buf$
- encrypt $x := TPKE.Enc(PK, proposed)$

// Step 2: Agreement on ciphertexts

- pass $x$ as input to ACS[$r$] //see Figure 4
- receive $\{v_j\}_{j \in S}$, where $S \subset [1..N]$, from ACS[$r$]

// Step 3: Decryption

- for each $j \in S$:
  - let $e_j := TPKE.DecShare(SK_i, v_j)$
  - multicast $\text{DEC}(r, j, i, e_j)$
  - wait to receive at least $f + 1$ messages of the form $\text{DEC}(r, j, k, e_{j,k})$
  - decode $y_j := TPKE.Dec(PK, \{(k, e_{j,k})\})$
- let $\text{block}_r := \text{sorted}(\bigcup_{j \in S}\{y_j\})$, such that $\text{block}_r$ is sorted in a canonical order (e.g., lexicographically)
- set $buf := buf - \text{block}_r$. 