Administrivia

- **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

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

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?

Lecture context

- **FLP:** “pick $\leq 2$ of Safety, Liveness, Fault-tolerance”
- So far have sacrificed liveness (Paxos, Raft, PBFT)
  - Want safety, fault-tolerance always
  - Settle for termination in practice (and avoid stuck states)
  - Partial and weak synchrony can help (e.g., PBFT)
- Two more ideas:
  - Remove asynchronous assumption entirely [Byzantine generals]
  - Remove deterministic assumption
- **Learning goals for today**
  - Two more ideas:
    - Commander
      - Why is it a problem?
      - FLP theorem
    - Some generals could be faulty, including commander
      - But loses safety – if any non-faulty node outputs $b$, decision will be $b$
      - 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 $\frac{1}{2}$
        - E.g., infinite rounds each with finite termination probability
      - Validity – if all correct nodes received input $b$, decision will be $b$

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 ($V_{G_0}$)
  - Round 1: Each other $G_i$ re-signs, broadcasts ($V_{G_i}G_0$)
  - 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 ($m$, $G_i$, $r + 1$ 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

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 $\frac{1}{2}$
    - E.g., infinite rounds each with finite termination probability
  - Validity – if all correct nodes received input $b$, decision will be $b$
Ben Or protocol [BenOr’83]

- ABA surviving $f$ faults for $N > 5f$ nodes

  Each node $i$ starts with input bit $x_i$, then executes:
  
  ```
  int x = x_i;
  for (round = 0; ++round) {
    if (round = 0) {
      broadcast <VOTE, round, x>
    } else {
      broadcast <COMMIT, round, x>
    }
  }
  ``

  - Why does this work?

Reliable broadcast (RBC) [Bracha]

- Sender $P_3$ has input $h$ to broadcast to $N > 3f$ nodes \{$P_i$\}
- Want: agreement, totality, validity [define these]

  Protocol
  1. $P_3$ 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 $h$

  - Why doesn’t RBC directly give us consensus?
    - Each node RBCs its input; take median (like Byz. generals)

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 (instance, round-number)
  - 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 $P_3$ has input $h$ to broadcast to $N > 3f$ nodes \{$P_i$\}
- Want: agreement, totality, validity [define these]

  Protocol
  1. $P_3$ 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 $h$

  - $N - f$ nodes includes majority of non-faulty nodes
    - READY from all non-faulty nodes has same $h$ $\implies$ agreement
    - If $P_3$ non-faulty, will all contain $P_3$’s input $h$ $\implies$ validity
  - If $2f + 1$ nodes send READY($h$), then $f + 1$ will be non-faulty
    - Those $f + 1$ will make all non-faulty nodes to broadcast READY($h$)
    - Since $N > 3f$, will get $2f + 1$ broadcasting READY($h$) $\implies$ totality

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 $P_3$ 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 \( P \) 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 \( VAL(h, b, s) \), broadcast \( ECHO(h, b, s) \)
  - \( s \) is share of message, \( b \) is proof that it is in hash tree with root \( h \)
  - Wait for \( N - f \) ECHO messages that permit reconstruction before sending \( READY(h) \) (guaranteed after \( 2f + 1 \) \( READY(h) \))

Asynchronous common subset (ACS)

- \( \{P_i\} \) get input, all output subset of inputs
  - Want: validity, agreement, totality
  - \( \{P_i\} \) get input, all output subset of inputs
while \( \text{fewer than } N - f \) RBCs have delivered a value
  & fewer than \( N - f \) ABA instances have output 1 \} \{ if \( (\text{RBC}_j \text{ delivers } v_{-j}) \)
  Supply 1 as input to \( \text{ABA}_j \)
\}
Supply 0 as input to any remaining ABAs
Output \( \{ v_{-j} | \text{ABA}_j \text{ output 1 } \} \) [waiting for RBCs if needed]
- Why does this ACS work?

Mostéfaoui ABA [Mostéfaoui’14]

let \( \text{est} = \text{input}_\text{value} \) // estimate of output value (0 or 1)
\( r = 0 \) // round number (integer)
\( \text{RBC}_\text{results}[] = \text{infinite list of empty bit sets} \)
\{ thread for(;;) \{
  \text{<EST, r, est>} <-> \text{RBC}_\text{receive}
  \text{add est} to \text{RBC}_\text{results}[r]
\}
for (int \( r = 0 ;; r++) \{
  \text{thread for RBC broadcast <EST, r, est>}
  \text{wait until RBC_results[r] !=}, \text{let } w \text{ be in RBC_results[r]}
  \text{multicast <AUX, i, r, w>}
  \text{receive AUXes from } N - f \text{ senders with } w \text{ values in RBC_results[r]}
  s < - \text{common_coin(r)} & 1 \text{ (low bit)}
  \text{if among } N - f \text{ received AUXes have both } w=0 \text{ and } w=1
  \text{est} = s
  \text{else if all have same value } w \{
    \text{if } w = w \text{ and haven’t output yet}
    \text{output(w) // but keep going}
  \text{est} = w
\}
\}
\}

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?

Asynchronous common subset (ACS)

- \( \{P_i\} \) get input, all output subset of inputs
  - Want: validity, agreement, totality
while \( \text{fewer than } N - f \) RBCs have delivered a value
  & fewer than \( N - f \) ABA instances have output 1 \} \{ if \( (\text{RBC}_j \text{ delivers } v_{-j}) \)
  Supply 1 as input to \( \text{ABA}_j \)
\}
Supply 0 as input to any remaining ABAs
Output \( \{ v_{-j} | \text{ABA}_j \text{ output 1 } \} \) [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 \( \text{ABA}_j = 1 \) at any non-faulty node, then RBC 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

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?
**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?**

  - Each node RBCs threshold encryption of $\lfloor B/N \rfloor$ transactions
  - Only decrypt after ACS complete
  - Threshold allows decryption even if sender fails

---

**Putting it all together (HoneyBadger)**

```
Algorithm HoneyBadgerBFT (for node $P_i$)
Let $B = \Omega(N^2 \log N)$ be the batch size parameter.
Let $PK$ be the public key received from TPKESetup (executed by a dealer), and let $SK_i$ be the secret key for $P_i$.
Let buf := \{\} be a FIFO queue of input transactions.

// Step 1: Random selection and encryption
\[ s := \text{TPKE.Enc}(PK, \text{proposed}) \]
// Step 2: Agreement on ciphertexts
\[ v_j := \text{TPKE.DecShare}(SK_i, v_j) \]
\[ \text{receive } (v_j)_{j \in S}, \text{ where } S \subseteq \{1, N\}, \text{ from ACS}\]

// Step 3: Decryption
\[ \text{for each } j \in S: \]
\[ \text{let } e_j := \text{TPKE.DecShare}((S, v_j)) \]
\[ \text{wait to receive at least } f + 1 \text{ messages of the form } \]
\[ \text{DEC}(i, j, a_j) \]
\[ \text{decode } y_j := \text{TPKE.Dec}(PK, (i, e_j, a_j)) \]
\[ \text{Let blocks, := sorted}(\{(j, y_j)\}), \text{ such that block, is sorted in a } \]
\[ \text{canonical order (e.g., lexicographically) } \]
\[ \text{set buf := buf + block, } \]
```