Lecture context

- **FLP: "pick ≤ 2 of Safety, Liveness, Fault-tolerance[1]"**
- **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**
  - Learn about randomized *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 deterministic, asynchronous protocol

Byzantine generals problem [Lamport'82]

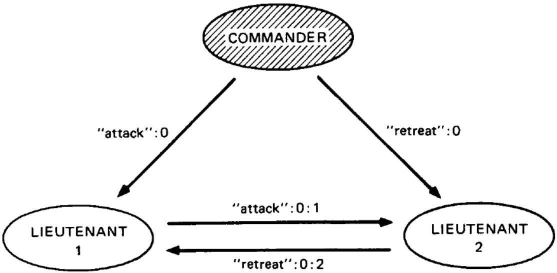


Fig. 5. Algorithm SM(1); the commander a traitor.

- **Commander G_0 sends a message to lieutenants $\{G_1, \dots, G_n\}$**
  - 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 signature, 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
- **Non-termination assumes network is *adversarial***
- **If were *random*, could have round termination probability**
  - Unfortunately, network typically can be controlled by adversary
  - But adversarial network can't predict randomness
  - So can we 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, none outputs $\neg b$
  - 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
    - Otherwise, okay to decide either 0 or 1

Ben Or protocol [BenOr'83]

function BenOr-ABA(i, x) ▷ i is local node id, x is input bit
  for $r \leftarrow 0; ; r \leftarrow r + 1$ **do**
    broadcast $\langle \text{VOTE}, i, r, x \rangle$
    await VOTE from $N - f$ distinct i ▷ including self
    if $\exists v$ s.t. more than $(N + f)/2$ VOTEs have $x = v$ **then**
      broadcast $\langle \text{COMMIT}, i, r, v \rangle$
    else
      broadcast $\langle \text{COMMIT}, i, r, \mathbf{?} \rangle$
    await COMMIT from $N - f$ distinct i ▷ including self
    if $\exists v \neq ?$ s.t. at least $f + 1$ COMMITs contain v **then**
      $x \leftarrow v$
      if more than $(N + f)/2$ COMMITs contain v **then**
        output v
    else
      $x \leftarrow$ random bit

- **Claim: BenOr-ABA survives f Byzantine faults for $N > 5f$ nodes**

Ben Or analysis

- **$> (N+f)/2$ nodes includes a majority of non-faulty nodes**
  - Majority of non-faulty nodes is $> (N-f)/2$ non-faulty nodes
  - Plus f faulty nodes means $> (N-f)/2 + f = (N+f)/2$
- **Hence, in a round, all non-faulty must COMMIT same $v \neq ?$**
  - But some or all non-faulty nodes may COMMIT ? instead
- **If receive $f+1$ COMMIT $v \neq ?$, know v must be correct**
  - After all, at most f of those nodes will be lying
- **Say you receive COMMIT v from C nodes where $C > (N+f)/2$**
  - Each other node will see at least $C - 2f$ COMMITs for v
    - because f of your C could double-vote, and another f could be slow
  - But since $N > 5f$, $C - 2f > (N+f)/2 - 2f > (5f+f)/2 - 2f = f$
  - So every other non-faulty node will see at least f+1 COMMITs for v
  - Means all other non-faulty nodes will terminate in next round
- **Say you don't see $f+1$ COMMITs and flip a coin**
  - Could luck out and have all non-faulty nodes flip same value
  - So protocol guaranteed to terminate eventually with probability 1!

Ben Or practicality

- **So why not use Ben Or instead of PBFT?**

Ben Or practicality

- **So why not use Ben Or instead of PBFT?**
  - Only agrees on one bit, not arbitrary operation
  - Exponential expected #rounds required when flipping coins
- **What if $N - f$ nodes got together and all flipped the same coin?**
  - Some honest nodes might see $f + 1$ COMMIT v_r, some not
  - But all who do will see the same v_r in round r
  - Let v'_r be the "common coin" flip
  - If $v'_r = v_r$, protocol terminates in round $r + 1$
- **Would it work to say rth coin flip is rth digit of π in binary?**

Ben Or practicality

- **So why not use Ben Or instead of PBFT?**
  - Only agrees on one bit, not arbitrary operation
  - Exponential expected #rounds required when flipping coins
- **What if $N - f$ nodes got together and all flipped the same coin?**
  - Some honest nodes might see $f + 1$ COMMIT v_r, some not
  - But all who do will see the same v_r in round r
  - Let v'_r be the "common coin" flip
  - If $v'_r = v_r$, protocol terminates in round $r + 1$
- **Would it work to say rth coin flip is rth digit of π in binary? No**
- **Problem: Adversary knows v'_r in advance and can influence v_r**
  - Arrange for $N - f - 1$ to see $f + 1$ COMMIT $\neg v'_r$ in round $r - 1$
  - Ensures $v_r \neq v'_r$, allows same manipulation for round $r + 1$
  - Never terminates so long as adversary is lucky in round 0
- **What if adversary doesn't know v'_r in advance?**

Common coin [Rabin'83]

- **Tool: t-of-N threshold cryptography**
  - Public key algorithm, using standard public key (e.g., RSA)
  - Private key broken into N shares, with t required to sign/decrypt
- **Tool: deterministic/unique digital signature schemes**
  - Only one possible signature per public key and message
  - E.g., RSA full-domain-hash, BLS. (Non-examples: Schnorr, DSA)
- **Idea: let coin $v'_r = \langle r \rangle_K \bmod 2$ for deterministic signature**
  - Private key K^{-1} split among agents with $(N - f)$-of-N threshold
  - Now v'_r unpredictable, but computable by any $N - f$ nodes
- **Limitation: setting up common coin requires trusted dealer**
  - Or can use fancy crypto, but requires *synchronous* protocol

Common coin Ben Or

function BenOrCC-ABA(i, x) ▷ i is local node id, x is input bit
  for $r \leftarrow 0; ; r \leftarrow r + 1$ **do**
    ⋮
    if $\exists v \neq ?$ s.t. at least $f + 1$ COMMITs contain v **then**
      $x \leftarrow v$
      if more than $(N + f)/2$ COMMITs contain v **then**
        output v
      CommonCoin(r) ▷ participate but discard result
    else
      $x \leftarrow$ CommonCoin(r) ▷ implicit private key share arg.

- **Note Rabin proposed a different trick for common coin**
  - If bad network knows you need $(N + f)/2$ votes to decide, can ensure some nodes see over, some under threshold
  - So use common coin to select threshold from $\{N/2, N - 2f\}$
  - Repeat R times, but only safe with probability $1 - 2^{-R}$

Reliable broadcast (RBC) [Bracha]

- **Sender P_S has input h to broadcast to $N > 3f$ nodes $\{P_i\}$**
- **Want:**
  - *agreement* – all non-faulty node outputs are identical
  - *totality* – all non-faulty nodes output a value or none terminate
  - *validity* – if P_S non-faulty, then all non-faulty nodes output h
- **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

RBC analysis

- **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
- $N - f$ **nodes includes majority of non-faulty nodes**
  - READY from all non-faulty nodes has same h $\Longrightarrow$ agreement
  - If P_S non-faulty, will all contain P_S's input $h \Longrightarrow$ 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) $\Longrightarrow$ totality

Refining RBC

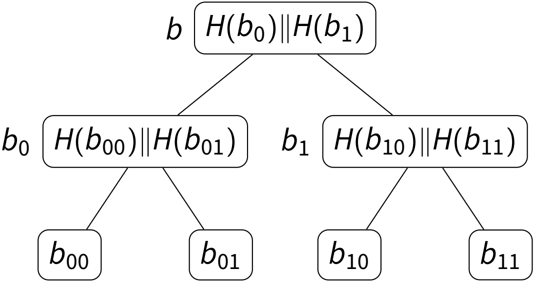
- **Why doesn't RBC directly give us consensus?**
  - Each node RBCs its input; take median (like Byz. generals)

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?**
- **Tool: Erasure coding**
  - Turns t-block msg into N blocks, such that *any* t encoded blocks are sufficient to reconstruct msg
  - Example: use interpolation on $(t-1)$-degree polynomial
- **Tool: Merkel tree**
  - Let $h = H(b)$
  - Can verify any b_{ij} from h and path in tree



Refining RBC (continued)

- **Change protocol to send** $\textbf{VAL}(h, b_i, s_i)$**, broadcast** $\textbf{ECHO}(h, b_i, s_i)$
  - 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** $\textbf{READY}(h)$
  - Guaranteed after $2f + 1$ READY(h)

- **Idea: use techniques from RBC to improve ABA**
  - Know an input b is valid if you received it
  - Also know b is valid if $f + 1$ nodes received it
  - Everyone will learn b is valid if $2f + 1$ nodes say it is
    - Even if f fail, $f + 1$ will continue to vouch for b

Mostéfaoui ABA [Mostéfaoui'14]

function MOSTÉFAOUI-ABA(i, x)
  for $r \leftarrow 0; ; r \leftarrow r + 1$ **do**
    $values \leftarrow \emptyset$ ▷ values everyone will consider valid
    broadcast $\langle \text{VALID}, i, r, x \rangle$
    when $\exists v$ s.t. received $\langle \text{VALID}, i', r, v \rangle$ from $f + 1$ distinct i'
      broadcast $\langle \text{VALID}, i, r, v \rangle$ if haven't already
    when $\exists v$ s.t. received $\langle \text{VALID}, i', r, v \rangle$ from $2f + 1$ distinct i'
      $values \leftarrow values \cup \{v\}$
    when $\exists w \in values$ and haven't sent VOTE yet
      broadcast $\langle \text{VOTE}, i, r, w \rangle$
    when received $N - f$ valid VOTEs (valid means $w \in values$)
      $s \leftarrow$ COMMONCOIN(r)
      if all $N - f$ valid VOTEs contain the same value b **then**
        $x \leftarrow b$
        if $b = s$ **then output** b
      else
        $x \leftarrow s$

Asynchronous common subset (ACS)

- **N nodes $\{P_i\}$ get input, all output subset of inputs. Want:**
  - *validity* – any non-faulty node output contains $N-2f$ non-faulty node inputs
  - *agreement* – if any non-faulty node outputs set V, all output same set V
  - *totality* – if $N-f$ non-faulty nodes get input, all non-faulty produce output

```
Node i submits v_i to RBC_i
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 output 1 }  [waiting for RBCs if needed]
```

- **Why does this ACS work?**

ACS continued

```
Node i submits v_i to RBC_i
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 output 1 }  [waiting for RBCs if needed]
```

- **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?

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?

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?**

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

Putting it all together (HoneyBadger)

Algorithm HoneyBadgerBFT (for node $\mathcal{P}_i$)

Let $B = \Omega(\lambda N^2 \log N)$ be the batch size parameter.
Let PK be the public key received from $\mathsf{TPKE.Setup}$ (executed by a dealer), and let SK_i be the secret key for $\mathcal{P}_i$.
Let buf $:= [\,]$ be a FIFO queue of input transactions.
Proceed in consecutive epochs numbered r:

// Step 1: Random selection and encryption

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

// Step 2: Agreement on ciphertexts

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

// Step 3: Decryption

- for each $j \in S$:

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

Discussion

- **Would you use HoneyBadgerBFT for a network file system like BFS?**

Discussion

- **Would you use HoneyBadgerBFT for a network file system like BFS?**
  - No - very high latency (10s of seconds) would give unusable performance
  - Also doesn’t take advantage of physical-layer multicast

- **Why use HoneyBadgerBFT instead of PBFT?**

Discussion

- **Would you use HoneyBadgerBFT for a network file system like BFS?**
  - No - very high latency (10s of seconds) would give unusable performance
  - Also doesn't take advantage of physical-layer multicast
- **Why use HoneyBadgerBFT instead of PBFT?**
  - High throughput with many replicas, big batch sizes
  - No need to worry about tuning timeouts