Byzantine Fault Tolerance Based Consensus for Blockchains

Yongge Wang

UNC Charlotte

Outline

References

The talk is based on the following two papers:

- Y. Wang: Byzantine Fault Tolerance in Partially Synchronous Networks
  `https://eprint.iacr.org/2019/1460`
- Y. Wang: Deterministic Blockchain BFT Protocol XP for Complete Asynchronous Networks
  `https://arxiv.org/abs/2005.04309`

Byzantine Agreement

“We imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement ...”. (Lamport, Shostak and Pease, 1980)

Byzantine Fault Tolerance (BFT)

- There are $n = 3t + 1$ participants and each participant starts with a bit.
- at most t traitors
- The goal is that all participants output the same bit

If all participants have direct communication channel, then this is TRUE. What happens if the network is NOT a COMPLETE graph?

PoS Blockchain and BFT

- In Proof of Stake (PoS) blockchain, there could be several qualified participants to generate the next block.
- This could lead to fork of the blockchain
- We have a BFT committee to determine which branch to take

Challenges for BFT in an Internet environment

- Internet is an open network. One may attack the communication link (e.g., DoS attacks) to make certain message not deliverable.
- Consequence: not all messages are delivered

Asynchronous and partial synchronous

- In an asynchronous network model, we do not make any assumption when the message will arrive. The only assumption that we will make is that: an honest participant will try to re-broadcast his/her message. So an honest participant will receive $2t + 1$ messages in the end (not assumption on the delivery time)
- In a partial synchronous network, there is an unknown GST (Global Stabilization Time). After GST, all messages are delivered within a known time period Δ.

BFT requirement

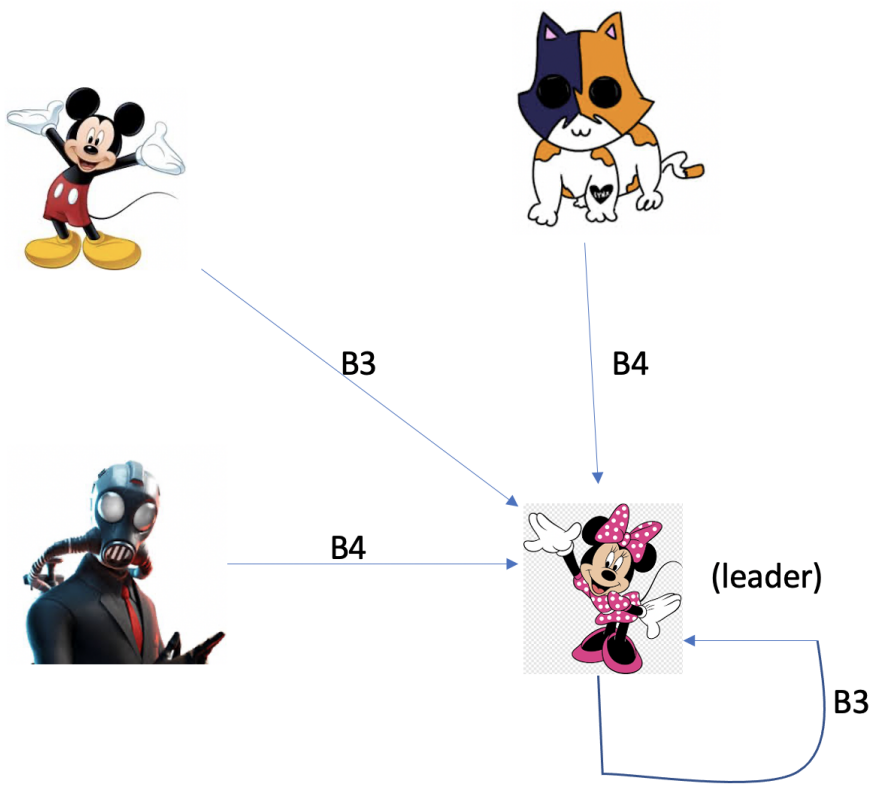
- **Safety**: If an honest participant decides on a value, then all other honest participants decides on the same value.
- **Liveness (termination)**: There exists a function $B(\cdot)$ such that all honest participants should decide on a value after the protocol runs at most $B(n)$ steps. It should be noted that $B(n)$ could be exponential in n. In this case, we should further assume that 2^n is significantly smaller than 2^κ where κ is the security parameter for the underlying authentication scheme.
- **Non-triviality (Validity)**: If all honest participants start the protocol with the same initial value, then all honest participants that decide must decide on this value.

An BFT Example — Based on BDLS for partial synchronous networks

- We have four participants
- Every one has a block (the bad guy may have more and send more to different people)
- They want to agree on a block. At the end of the protocol, all honest guys should have the same block

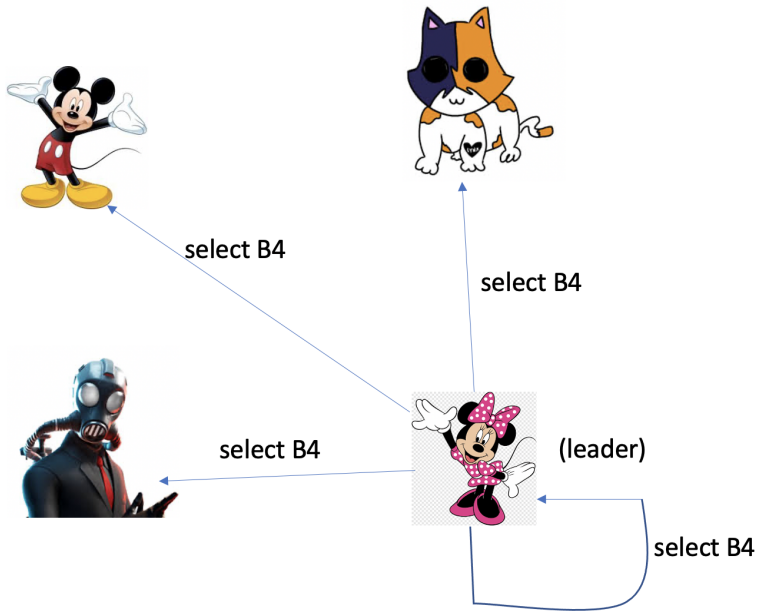
An BFT Example – BDLS (1)

- Leader: Minnie
- Mickey $\longrightarrow$ B3, BabyMeowscles $\longrightarrow$ B4, ChaosAgent$\longrightarrow$B4, and Minnie $\longrightarrow$ B3.



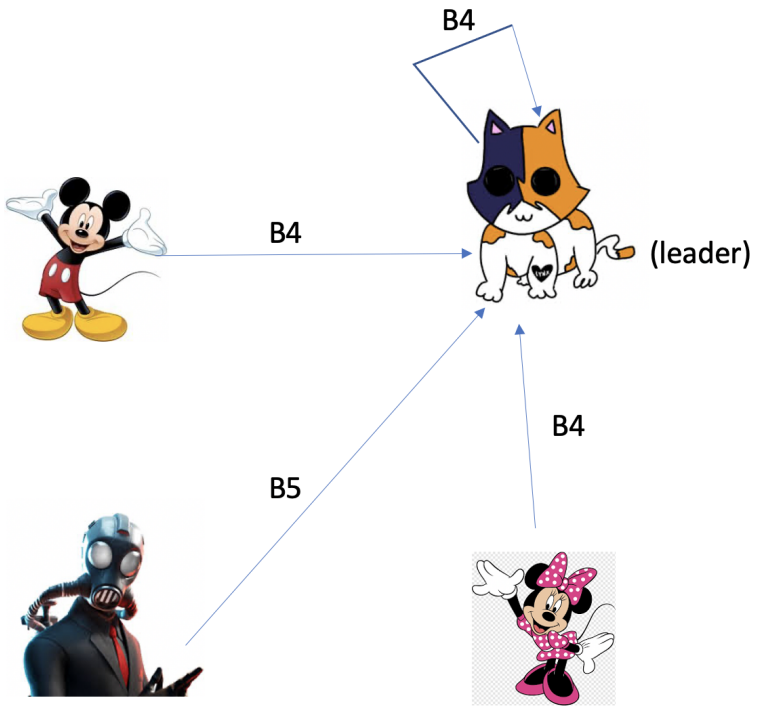
An BFT Example – BDLS (2)

- In order for Minnie to decide, she needs to receive an identical block from three participants. But she received two B3 and two B4, she cannot decide.
- Minnie selects best learned B4 and sends “select B4” to all. Each updates his block to the best learned one
- Move to the next round with BabyMeowscles as leader



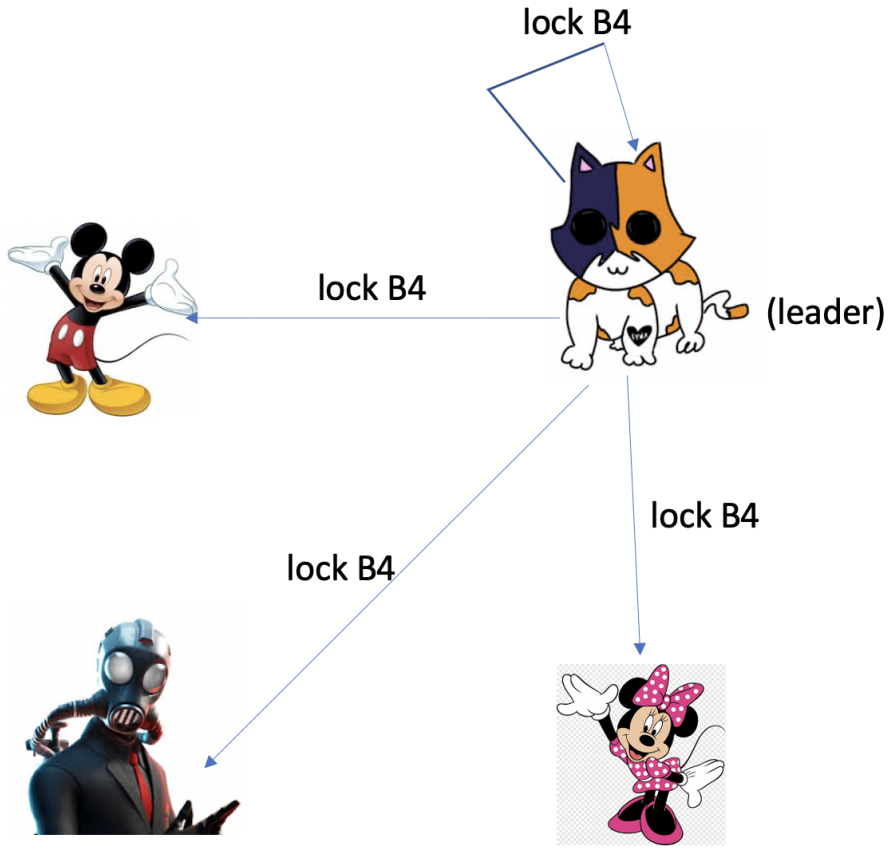
An BFT Example – BDLS (3)

- leader: abyMeowscles
- Mickey $\longrightarrow$ B4, BabyMeowscles $\longrightarrow$ B4, Minnie$\longrightarrow$ B4, and ChaosAgent $\longrightarrow$ B5
- By receiving 3 identical B4, BabyMeowscles is ready to decide B4. he first needs to lock B4



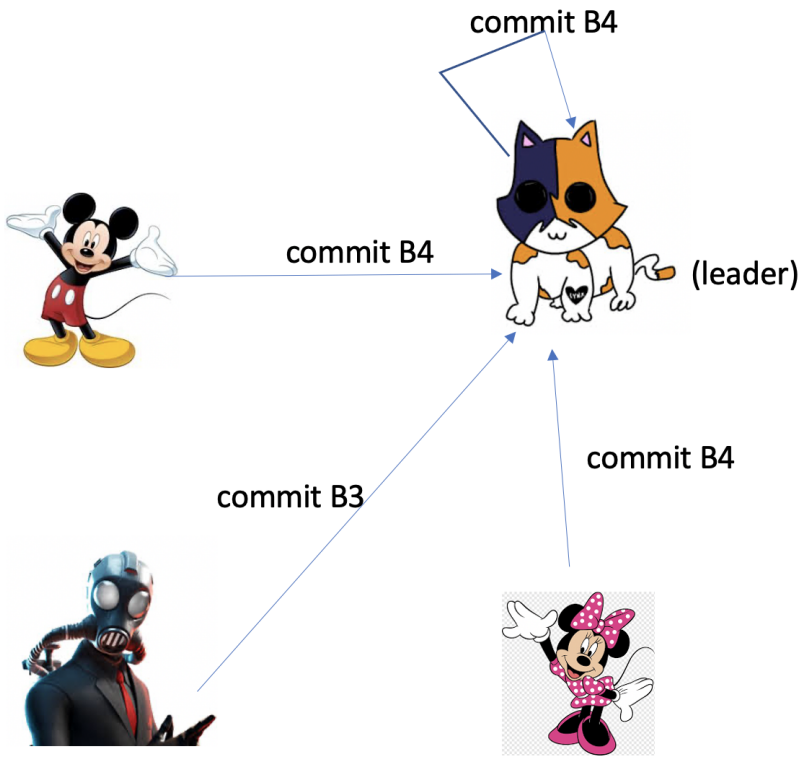
An BFT Example – BDLS (4)

- BabyMeowscles sends the “lock B4” instruction to all participants



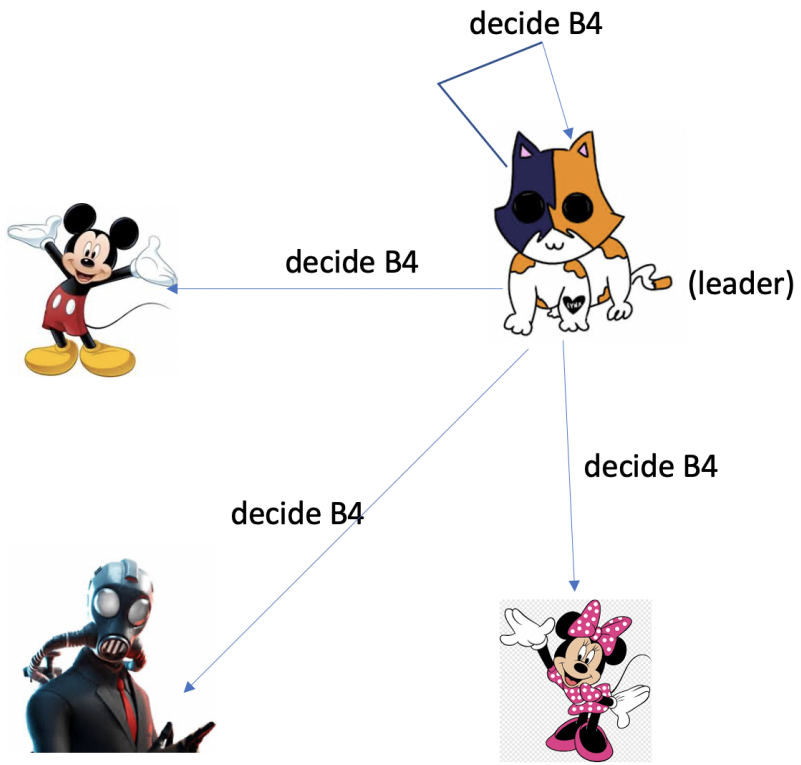
An BFT Example – BDLS (5)

- Every one except ChaosAgent send the “commit B4” message to the leader BabyMeowscles
- By receiving three identical commit message for the block B4, BabyMeowscles is ready to decide for an agreement



An BFT Example – BDLS (6)

- BabyMeowscles sends “decide B4” command to all
- Each one decide after receiving “decide B4” and relay the decision message to all.
- BDLS protocol reached an agreement on block B4

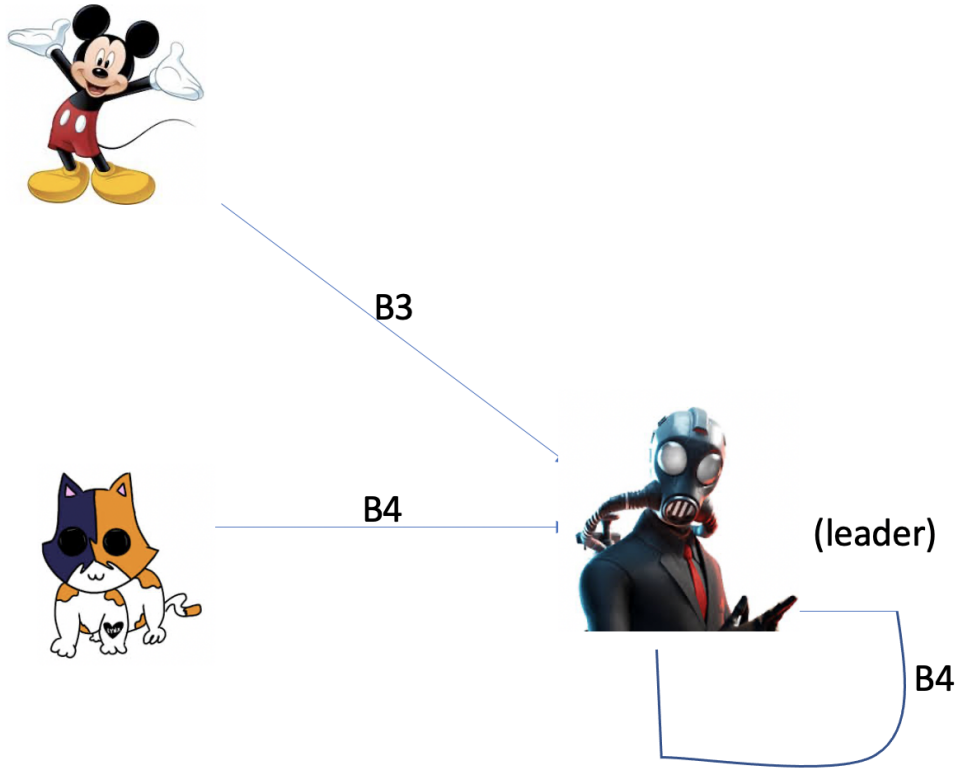


An BFT Example – Impossibility

- there are three participants and one bad guy, no agreement can be achieved

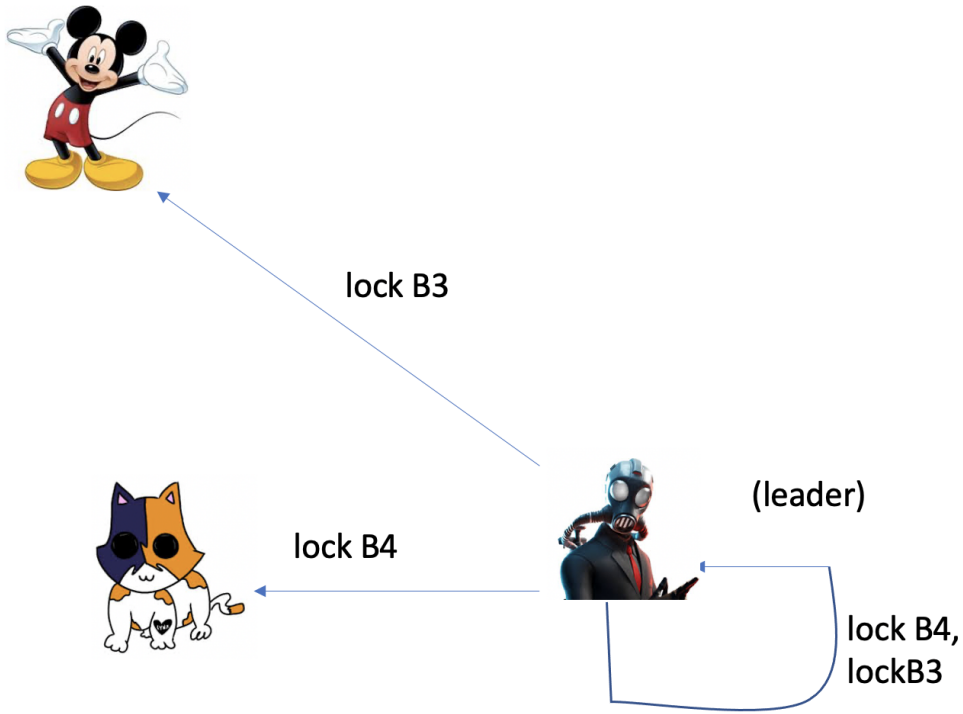
An BFT Example – Impossibility (1)

- At start: Mickey holds B3, BabyMeowscles holds B4, ChaosAgent holds B4
- Leader: ChaosAgent who starts his cheating process



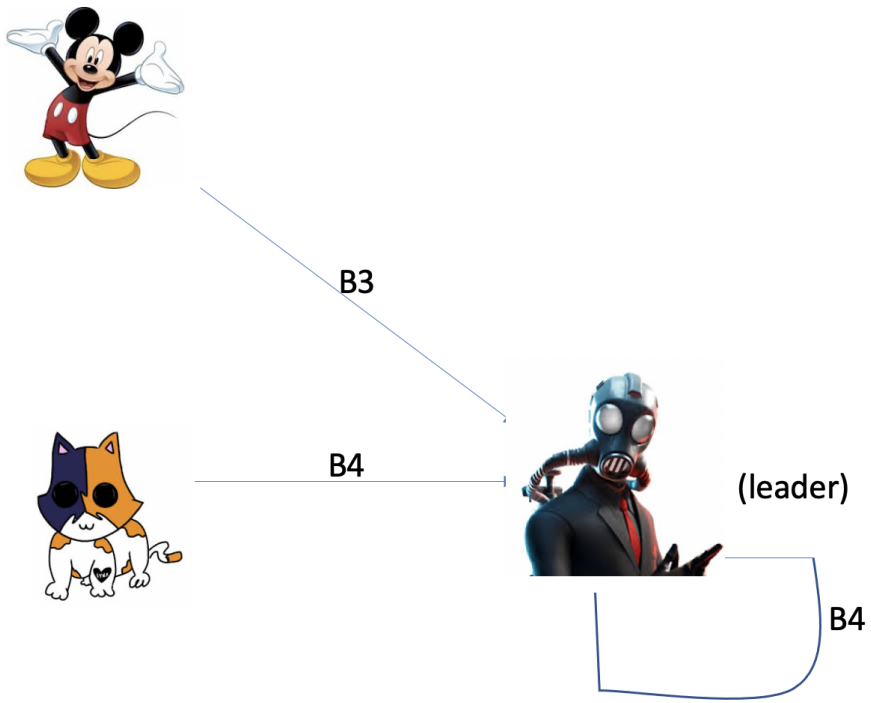
An BFT Example – Impossibility (2)

- ChaosAgent sends “lock B3” to Mickey and “lock B4” to BabyMeowscles
- ChaosAgent can send both “lock B3” and “lock B4” to himself



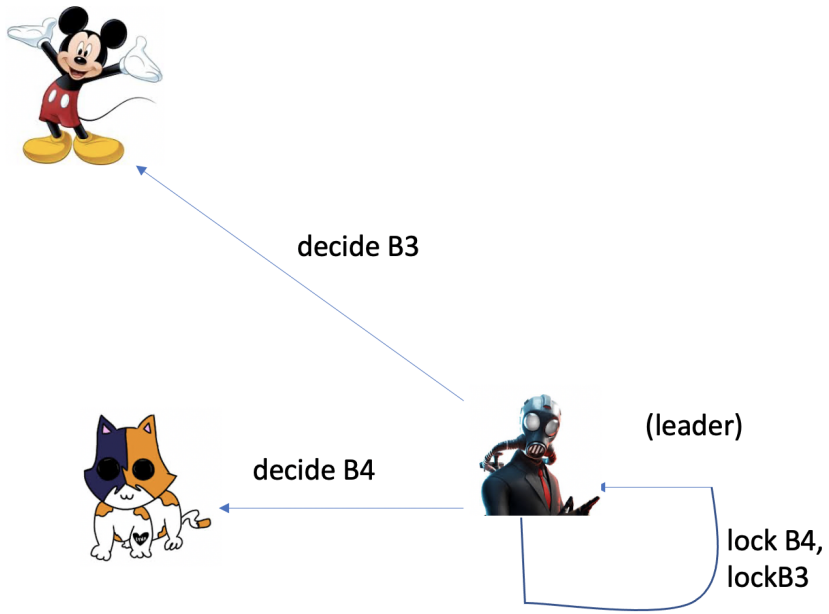
An BFT Example – Impossibility (3)

- Now Mickey sends “commit B3” to ChaosAgent , and BabyMeowscles sends “commit B4” to ChaosAgent
- ChaosAgent is ready to let Mickey and BabyMeowscles to decide on different values now



An BFT Example – Impossibility (4)

- ChaosAgent sends “decide B3” to Mickey and “decide B4” to BabyMeowscles
- Now Mickey decides on the block B3 and BabyMeowscles decides the block on B4
- Note that Mickey and BabyMeowscles are both good guys but they have different block now



Ethereum's CBC Casper the Friendly Binary Consensus (FBC)

- Let $\mathcal{M}_{i,s}$ be the collection of valid messages that P_i has received from all participants (including himself) from steps $0, \cdots, s-1$. P_i determines whether a consensus has been achieved. If a consensus has not been achieved yet, P_i sends the message

$$m_{i,s} = \langle P_i, e_{i,s}, \mathcal{M}_{i,s} \rangle \tag{1}$$

to all participants where $e_{i,s}$ is P_i's estimated consensus value based on the received message set $\mathcal{M}_{i,s}$.

Ethereum's CBC-FBC

For a message $m = \langle P_i, e_{i,s}, \mathcal{M}_{i,s} \rangle$, let $J(m) = \mathcal{M}_{i,s}$. For two messages m_1, m_2, we write $m_1 \prec m_2$ if m_2 depends on m_1. That is, there is a sequence of messages $m'_1, \cdots, m'_v$ such that

$$\begin{aligned} m_1 &\in J(m'_1) \\ m'_1 &\in J(m'_2) \\ &\cdots \\ m'_v &\in J(m_2) \end{aligned}$$

For a message m and a message set $\mathcal{M} = \{m_1, \cdots, m_v\}$, we say that $m \prec \mathcal{M}$ if $m \in \mathcal{M}$ or $m \prec m_j$ for some $j = 1, \cdots, v$. The *latest message* $m = L(P_i, \mathcal{M})$ by a participant P_i in a message set $\mathcal{M}$ is a message $m \prec \mathcal{M}$ satisfying the following condition:

- There does not exist another message $m' \prec \mathcal{M}$ sent by participant P_i with $m \prec m'$.

Ethereum's CBC-FBC

For a binary value $b \in \{0, 1\}$ and a message set $\mathcal{M}$, the score of a binary estimate for b is defined as the number of non-equivocating participants P_i whose latest message voted for b. That is,

$$\texttt{score}(b, \mathcal{M}) = \sum_{L(P_i, \mathcal{M}) = (P_i, b, *)} \lambda(P_i, \mathcal{M}) \tag{2}$$

where

$$\lambda(P_i, \mathcal{M}) = \begin{cases} 0 & \text{if } P_i \text{ equivocates in } \mathcal{M}, \\ 1 & \text{otherwise.} \end{cases}$$

Ethereum's CBC-FBC

To estimate consensus value:

Now we are ready to define P_i's estimated consensus value $e_{i,s}$ based on the received message set $\mathcal{M}_{i,s}$ as follows:

$$e_{i,s} = \begin{cases} 0 & \text{if } \texttt{score}(0, \mathcal{M}_{i,s}) > \texttt{score}(1, \mathcal{M}_{i,s}) \\ 1 & \text{if } \texttt{score}(1, \mathcal{M}_{i,s}) > \texttt{score}(0, \mathcal{M}_{i,s}) \\ b & \text{otherwise, where } b \text{ is coin-flip output} \end{cases} \tag{3}$$

Ethereum's CBC-FBC

To infer consensus achievement:

For a protocol execution, it is required that for all i, s, the number of equivocating participants in $\mathcal{M}_{i,s}$ is at most t. A participant P_i determines that a consensus has been achieved at step s with the received message set $\mathcal{M}_{i,s}$ if there exists $b \in \{0, 1\}$ such that

$$\forall s' > s : \texttt{score}(b, \mathcal{M}_{i,s'}) > \texttt{score}(1 - b, \mathcal{M}_{i,s'}). \quad (4)$$

Waiting in asynchronous networks

- A participant P_i should wait for at least $n - t + |I(\mathcal{M}_{i,s})|$ valid messages $m_{j,s-1}$ from other participants before he can broadcast his step s message $m_{i,s}$.
- In case that a participant P_i receives $n - t + |I(\mathcal{M}_{i,s})|$ valid messages $m_{j,s-1}$ from other participants before he could post his step $s - 1$ message, he should wait until he finishes sending his step $s - 1$ message.
- After a participant P_i posts his step s protocol message, it should discard all messages from steps $s - 1$ or early except decision messages that we will describe later.

CBC BFT with Bracha's reliable broadcast primitive

1. P_i broadcasts $\langle P_i, x_i, \mathcal{M}_{i,s,0} \rangle$ where $\mathcal{M}_{i,s,0}$ is what P_i has received during step $s-1$. Then P_i waits until $|\mathcal{M}_{i,s,1}| \geq n-t$ and computes estimate $e_{i,s}$
2. P_i broadcasts $\langle P_i, e_{i,s}, \mathcal{M}_{i,s,1} \rangle$ and waits until $|\mathcal{M}_{i,s,2}| \geq n-t$. If $\texttt{score}(b, \mathcal{M}_{i,s,2}) > \frac{n}{2}$ for some b, then $e'_{i,s} = b$ otherwise $e'_{i,s} = \perp$.
3. P_i broadcasts $\langle P_i, e'_{i,s}, \mathcal{M}_{i,s,2} \rangle$ and waits until $|\mathcal{M}_{i,s,3}| \geq n-t$. P_i distinguishes 3 cases:
   - If $\texttt{score}(b, \mathcal{M}_{i,s,2}) > 2t+1$ for some $b \in \{0,1\}$, then P_i decides on b and broadcasts his decision.
   - If $\texttt{score}(b, \mathcal{M}_{i,s,2}) > t+1$ for some $b \in \{0,1\}$, then P_i lets $x_i = b$ and moves to step $s+1$.
   - Otherwise, P_i flips a coin and let x_i to be coin-flip outcome. P_i moves to step $s+1$.

Deterministic leaderless BFT (1)

- **lock**: If $s = 0$, then let $B = x_i$. Otherwise, if $s > 0$, select $n - t$ valid commit-votes from step $s - 1$ and let

$$B = \begin{cases} B' & P_i \text{ receives a commit-vote for } B' \text{ in step } s-1 \\ x_i & \text{otherwise} \end{cases} \tag{5}$$

  Then P_i sends the following message to all participants.

$$\langle P_i, \texttt{lock}, s, B, \texttt{justification} \rangle \tag{6}$$

  where justification consists of a list of messages to show that his selection of the value B is justified.

Deterministic leaderless BFT (2)

- **commit**: P_i collects $n - t$ valid and justified step-s `lock` messages (6). If there is any candidate block B' from these lock messages such that $x_i \prec B'$, then P_i lets $x_i = B'$. Furthermore, P_i lets

$$\bar{B} = \begin{cases} B & \text{if there are } n - t \text{ locks for } B \\ \perp & \text{otherwise} \end{cases} \tag{7}$$

  Then P_i sends the following message to all participants

$$\langle P_i, \texttt{commit}, s, \bar{B}, x_i, \texttt{justification} \rangle \tag{8}$$

  where justification consists of a list of messages to show that his selection of the value $\bar{B}$ is justified.

Deterministic leaderless BFT (3)

- **check-for-decision**: Collect $n - t$ properly justified commit votes (8) of step s. If there is any commit vote $\langle P_j, \texttt{commit}, s, *, x_j, \texttt{justification}\rangle$ with $x_i \prec x_j$, then P_i lets $x_i = x_j$. Furthermore, if these are $n - t$ commit-votes for a block $\bar{B}$, then P_i decides the block $\bar{B}$ and continues for one more step (up to `commit` sub-step). Otherwise, simply proceed.

BFT in partial synchronous networks: BDLS (1)

1. Each participant P_j (including P_i) sends the signed message $(\langle h, r\rangle_j, \langle h, r, B'_j\rangle_j)$ to the leader P_i where $B'_j \in \text{BLOCK}_j$ is the maximal acceptable candidate block for P_j. The message $\langle h, r\rangle_j$ is considered as a round-change message. After sending the round-change message, P_j will not accept messages except a "decide" message for round $r' < r$ anymore.

BFT in partial synchronous networks: BDLS (2)

1. If P_i receives at least $2t + 1$ round-change messages (including himself), it enters round r. In these round-change messages, if there are at least $2t + 1$ signed messages from $2t + 1$ participants with the same candidate block $B' \neq NULL$, then P_i broadcasts the following signed message (9) to all participants

$$\langle \text{lock}, h, r, B', \text{proof} \rangle_i \tag{9}$$

If P_i does not receive such a block B', then P_i adds all received candidate blocks to its local variable BLOCK_i and broadcasts $\langle \text{select}, h, r, B'', \text{proof} \rangle$ where B'' is the candidate block $B'' = \max\{B : B \in \text{BLOCK}_i\}$.

BFT in partial synchronous networks: BDLS (3)

1. If a participant P_j (including P_i) receives a valid $\langle \text{select}, h, r, B'', \text{proof} \rangle$ from P_i during Step 1, then it adds B'' to its BLOCK_j. If a participant P_j (including P_i) receives a valid message $\langle \text{lock}, h, r, B', \text{proof} \rangle_i$ from P_i in Step 1, then it does the following:
   1. releases any potential lock on B' from previous round, but does not release locks on any other potential candidate blocks
   2. locks the candidate block B' by recording the valid lock (9)
   3. sends the following signed commit message to the leader P_i.

$$\langle \text{commit}, h, r, B' \rangle_j. \tag{10}$$

BFT in partial synchronous networks: BDLS (4)

1. If P_i receives at least $2t + 1$ commit messages (10), then P_i *decides* on the value B' and broadcasts the following decide message to all participants

$$\langle \text{decide}, h, r, B', \text{proof} \rangle_i. \tag{11}$$

2. If a participant P_j (including P_i) receives a decide message (11) or from its neighbor, it decides on the block B' for B^h. Otherwise, it goes to the following lock-release step:
   - (*lock-release*) If a participant P_j (including P_i) has some locked values, it broadcasts all of its locked values. A participant releases its lock on a value $\langle \text{lock}, h, r'', B'', \text{proof} \rangle_{i''}$ if it receives a lock $\langle \text{lock}, h, r', B', \text{proof} \rangle_{i'}$ with $r' \geq r''$ and $B' \neq B''$.
   - Move to the next round $r + 1$.

Thanks!

Q&A?