Securing Proof-of-Work Ledgers via Checkpointing

Dimitris Karakostas, Aggelos Kiayias

Bitcoin’s novelties

- Hash chain +
- Proof-of-Work +
- Incentives for participation

Bitcoin’s novelties

- Hash chain +
- Proof-of-Work +
- Incentives for participation

<u>Distributed ledger</u>

↓

<u>Open (decentralised) consensus</u>

Proof-of-Work

- A *compute cycle* is one identity
- Limit the amount of identities per person
    - Cannot create more identities than CPU cycles one controls
    - Sybil protection

Proof-of-Work

- A *compute cycle* is one identity
- Limit the amount of identities per person
    - Cannot create more identities than CPU cycles one controls
    - Sybil protection
- Core security assumption: **<u>50%+1 CPU cycles are honest</u>**

51% attacks are real

zencash

ethereum
classic

BITCOINGOLD

pied
piper

Overview

- How can checkpoints secure an insecure ledger?
    - Checkpointing ideal functionality
    - Security guarantees
    - Ethereum Classic analysis
    - The protocol that realizes checkpointing functionality
- Distributed checkpointing prototype implementation
- Timestamping: decentralizing checkpoints

Our goals

- Secure a ledger *temporarily* against 51% attacks
- Avoid trivializing the ledger maintenance
- Minimize storage/time overhead

Core idea

- Introduce an external set of parties to guarantee security

Preliminaries

- Fixed number of parties (n)
- Round-based execution
- All messages are delivered by the end of a round (*synchronous*)
- Block size is unlimited

Preliminaries (cont.)

- Each party has q queries to a random oracle (*hashing power*)
- Each query is succesful with probability p
- The adversary $\mathcal{A}$:
  - controls t parties (equiv. $\mu_{\mathcal{A}} = t/n$ hashing power)
  - adaptive: corrupts parties on the fly
  - rushing: decides strategy after (possibly) delaying honest messages

Ledger properties

- **Stable transaction τ**: each honest party reports τ in the same position in the ledger

- **Persistence**: a transaction in a block at least k blocks away from the ledger's head is stable

- **Liveness**: a transaction which is continuously provided to the parties becomes stable after at most u rounds

Checkpointing functionality

- The *ideal* definition of checkpoints
- An omnipresent entity
- Expresses the needed security properties

Checkpointing functionality

- The *ideal* definition of checkpoints
- An omnipresent entity
- Expresses the needed security properties

Functionality $\mathcal{F}_{\mathsf{Checkpoint}}$

$\mathcal{F}_{\mathsf{Checkpoint}}$ interacts with a set of parties $\mathbb{V}$ and holds the local chain C and the checkpoint chain C_c, both initially set to ϵ. It is parameterized by k_c, which defines the number of blocks between two consecutive checkpoints, and the $\mathsf{maxvalid}(\cdot,\cdot)$ algorithm.

Upon receiving $(\textsc{CandidateCheckpoint}, C')$ from a party $\mathcal{V}$, forward it to $\mathcal{A}$. Upon receiving $(\textsc{CandidateCheckpoint}, C')$ from $\mathcal{A}$, if $C_c \prec C'$ set $C := \mathsf{maxvalid}(C, C')$. Next, if $|C \setminus C_c| = k_c$ compute a list R of $|\mathbb{V}|$ random values as $r_j \xleftarrow{\$} \{0,1\}^\omega$ and send $(\textsc{Nonce}, R)$ to $\mathcal{A}$. Upon receiving from $\mathcal{A}$ a response $(\textsc{Nonce}, R')$, such that R' is a list of at least $\frac{|\mathbb{V}|}{2}$ values from R, pick a value $r_i \in R'$, return $(\textsc{Checkpoint}, \mathit{tail}(C)||r_i)$ to $\mathcal{V}$ and set $C := C_c := C||r_i$.

Security of the checkpointed ledger

Persistence

(a transaction in a block at least **k** blocks away from the ledger's head is stable)

Persistence

- *k (persistence parameter) $\geq k_c$ (checkpoint interval)*
- At least one in the last k blocks is a checkpoint
- Checkpoints cannot be reverted
- All blocks up to the last checkpoint are stable

Security of the checkpointed ledger

Liveness

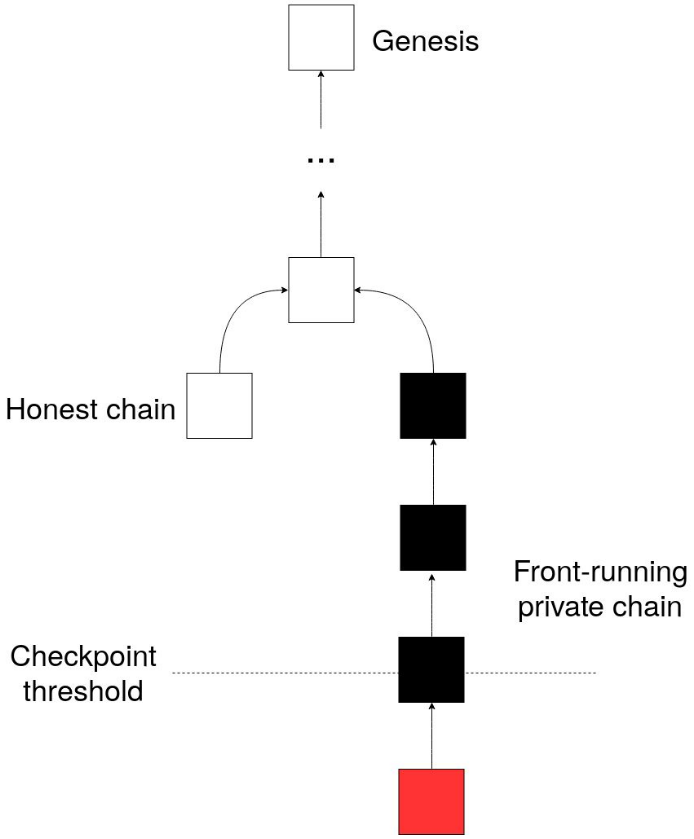
(a transaction which is continuously provided to the parties becomes stable after at most **<u>u</u>** rounds)

Liveness

- If an honest block B gets checkpointed *after* a transaction τ is created, then τ becomes stable
  - Proof: if τ is not in any block prior to B, then B will include it (because honest parties include all unpublished transactions and blocks are unlimited)

- Creating checkpoints is <u>not enough</u>; they need to be put in the chain

Front-running: An attack against liveness



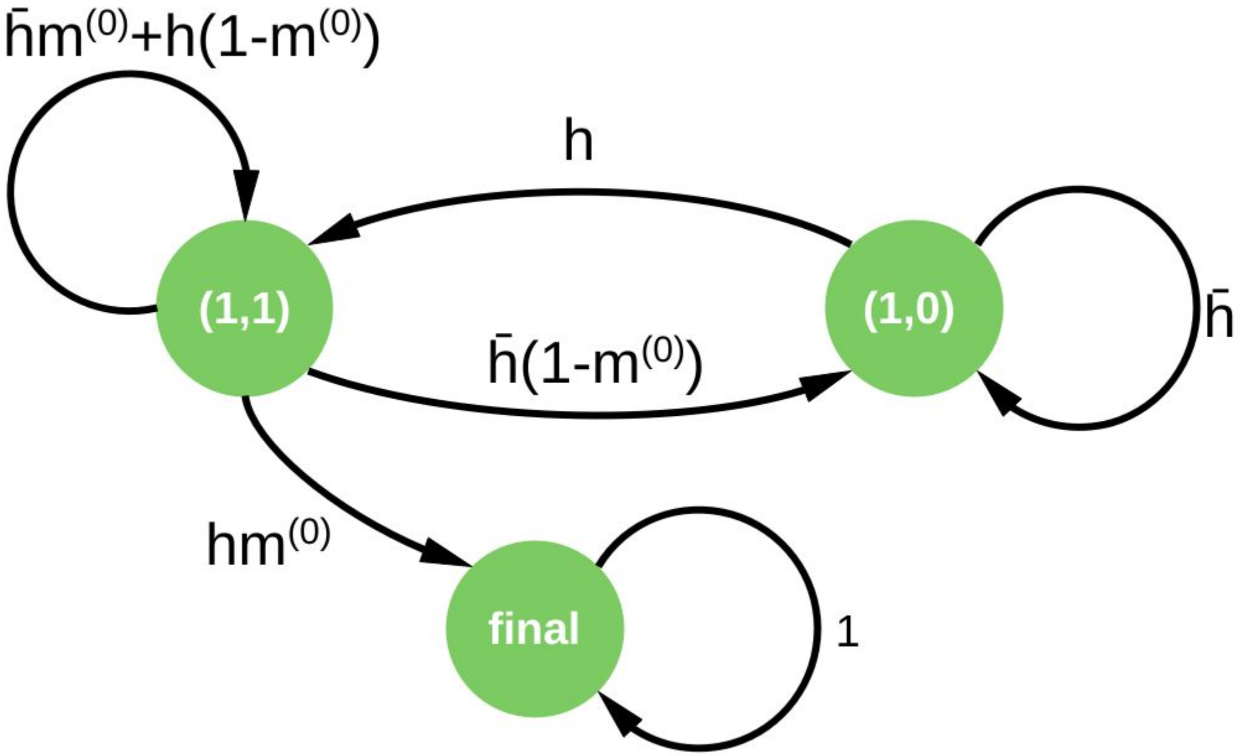
Liveness analysis

- Separate the honest from the adversarial parties
- Argue about security wrt. honest parties (regardless of adversarial strategy)
- Stochastic Markov chain for protocol execution modelling

Liveness Markov chain

- Each state is identified by (i, j):
  - i: the number of blocks an honest party <u>needs to produce</u> to reach the next checkpoint
  - j: the number of blocks the adversary <u>necessarily</u> needs to produce to reach the next checkpoint
- Random variables:
  - H: if at least one honest party produces a block at a given round, then $H = 1$, else $H = 0$
  - $M^{(i)}$: if all adversarial parties produce *i* blocks at a given round, then $M^{(i)} = 1$, else $M^{(i)} = 0$
- Expectations:
  - $E(H) = h = 1 - (1-p)^{q(n-t)}$
  - $E(M^{(i)}) = m^{(i)} = \binom{qt}{i} \cdot p^i \cdot (1-p)^{qt-i}$
- Transition probabilities ($b \geq 0$):
  - To $(i, j - b)$: $(1 - h) \cdot m^{(b)}$
  - To $(i - 1, j - b)$: $h \cdot m^{(b)}$

Liveness Markov chain (k_c = 1)

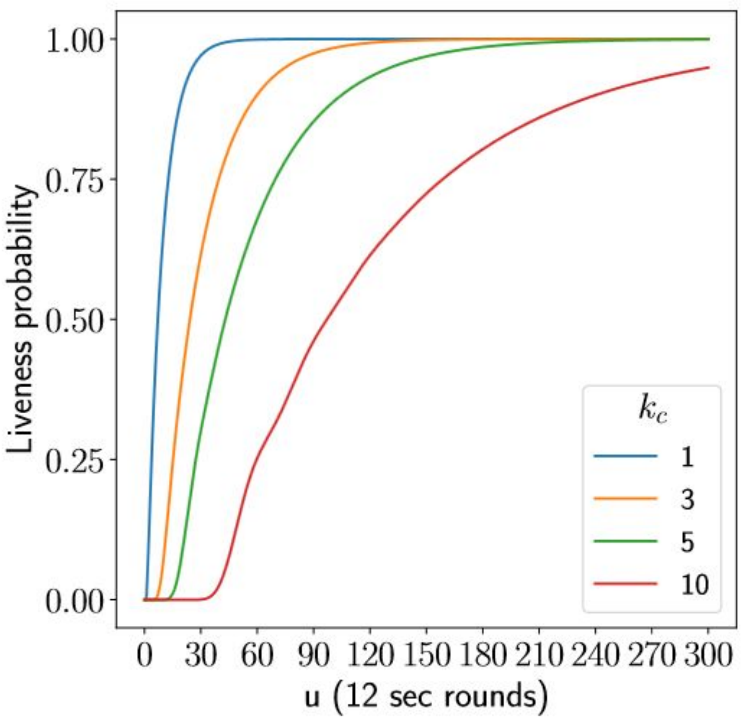


Markov chain properties

- Stochastic transition matrix: matrix that defines the transition probabilities between two states
- Canonical form: $M = \begin{pmatrix} Q & R \\ \mathbb{O} & I_r \end{pmatrix}$ (Q: transition states, R: absorption states)
- Probability of transition from s_i to s_j after u rounds: ij-th column of M^u
- Expected number of steps before absorption: $t = \lceil \sum_{j=0}^{t} N_{ij} \rceil, (I - Q)^{-1} = N$

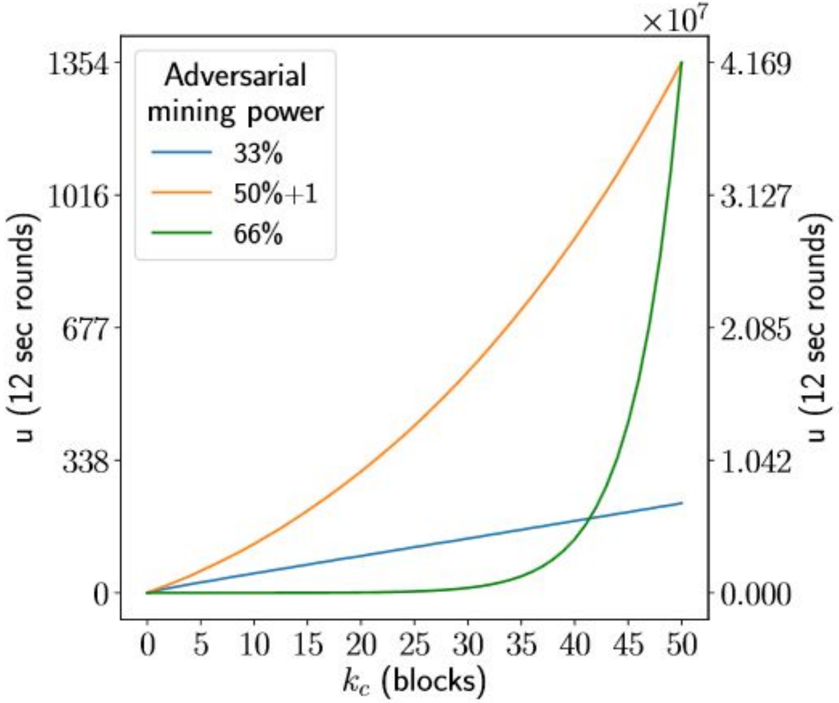
Liveness of a checkpointed Ethereum Classic

Liveness probability for 51% adversary



Liveness of a checkpointed Ethereum Classic

Expected number of steps before absorption



The checkpointing protocol

- Parameterized by a *fail-stop* protocol π_{fs}
- Every k_c blocks:
  - Pick a random nonce (eg. randomized signature)
  - Run π_{fs} to agree on checkpoint
  - Append nonce to chosen block

The checkpointing protocol

Protocol $\pi_{\mathsf{Checkpoint}}$

A checkpointing party which runs $\pi_{\mathsf{Checkpoint}}$ is parameterized by the list $\mathbb{V}$ of n checkpointing parties, a (fail-stop) consensus protocol π_{FS}, a validation predicate $\mathsf{Validate}$, the function $\mathsf{maxvalid}$, and k_c. It keeps a local checkpointed block, B_c, initially set to ϵ.

Upon receiving $(\textsc{CandidateCheckpoint}, C')$ from a party $\mathcal{V}$, check:

- $\exists i : C'[i] = B_c$ (i.e. *if C' extends the checkpoint*);
- $\mathsf{Validate}(C') = 1$ (i.e. *if C' is valid*);
- $|C'| - i = k_c$ (i.e. *if C' is long enough*).

If all hold do:

1. pick $r_j \xleftarrow{\$} \{0,1\}^{\omega}$;
2. pick input $\langle C', r_j \rangle$ for the protocol π_{FS};
3. execute π_{FS} with the parties in $\mathbb{V}$ to agree on an input $\langle C', r' \rangle$, such that $\forall \langle \hat{C}, \hat{r} \rangle \in \mathbb{I} : \mathsf{maxvalid}(C', \hat{C}) = C'$ with $\mathbb{I}$ the set of inputs, i.e. choose the output according to $\mathsf{maxvalid}$;
4. set $B_c := \mathit{tail}(C') || r'$.

Finally, return $(\textsc{Checkpoint}, B_c)$ to $\mathcal{V}$.

Proof strategy

- Show that ideal and real worlds are indistinguishable

Functionality $\mathcal{F}_{\mathsf{Checkpoint}}$

$\mathcal{F}_{\mathsf{Checkpoint}}$ interacts with a set of parties $\mathbb{V}$ and holds the local chain C and the checkpoint chain C_c, both initially set to ϵ. It is parameterized by k_c, which defines the number of blocks between two consecutive checkpoints, and the $\mathsf{maxvalid}(\cdot,\cdot)$ algorithm.

Upon receiving $(\textsc{CandidateCheckpoint}, C')$ from a party $\mathcal{V}$, forward it to $\mathcal{A}$. Upon receiving $(\textsc{CandidateCheckpoint}, C')$ from $\mathcal{A}$, if $C_c \prec C'$ set $C := \mathsf{maxvalid}(C, C')$. Next, if $|C \setminus C_c| = k_c$ compute a list R of $|\mathbb{V}|$ random values as $r_j \xleftarrow{\$} \{0,1\}^\omega$ and send $(\textsc{Nonce}, R)$ to $\mathcal{A}$. Upon receiving from $\mathcal{A}$ a response $(\textsc{Nonce}, R')$, such that R' is a list of at least $\frac{|\mathbb{V}|}{2}$ values from R, pick a value $r_i \in R'$, return $(\textsc{Checkpoint}, \mathit{tail}(C)||r_i)$ to $\mathcal{V}$ and set $C := C_c := C||r_i$.

$\approx$

Protocol $\pi_{\mathsf{Checkpoint}}$

A checkpointing party which runs $\pi_{\mathsf{Checkpoint}}$ is parameterized by the list $\mathbb{V}$ of n checkpointing parties, a (fail-stop) consensus protocol π_{FS}, a validation predicate $\mathsf{Validate}$, the function $\mathsf{maxvalid}$, and k_c. It keeps a local checkpointed block, B_c, initially set to ϵ.

Upon receiving $(\textsc{CandidateCheckpoint}, C')$ from a party $\mathcal{V}$, check:

- $\exists i : C'[i] = B_c$ (i.e. *if C' extends the checkpoint*);
- $\mathsf{Validate}(C') = 1$ (i.e. *if C' is valid*);
- $|C'| - i = k_c$ (i.e. *if C' is long enough*).

If all hold do:

1. pick $r_j \xleftarrow{\$} \{0,1\}^\omega$;
2. pick input $\langle C', r_j \rangle$ for the protocol π_{FS};
3. execute π_{FS} with the parties in $\mathbb{V}$ to agree on an input $\langle C', r' \rangle$, such that $\forall \langle \hat{C}, \hat{r} \rangle \in \mathbb{I} : \mathsf{maxvalid}(C', \hat{C}) = C'$ with $\mathbb{I}$ the set of inputs, i.e. choose the output according to $\mathsf{maxvalid}$;
4. set $B_c := \mathit{tail}(C')||r'$.

Finally, return $(\textsc{Checkpoint}, B_c)$ to $\mathcal{V}$.

Chain decision using checkpoints

- Every k_c blocks, send the last block to checkpoint authority
- Retrieve checkpoint, append it to the chain, and then keep mining

Protocol $\pi_{\mathsf{CheckpointMiningRes}}$

A party which runs $\pi_{\mathsf{CheckpointMiningRes}}$ is parameterized by maxvalid, the n checkpointing parties $\mathbb{V}$ which run $\pi_{\mathsf{Checkpoint}}$, and k_c. It keeps a local chain C and the checkpoint index i_c, initially set to ϵ and 0.

Upon receiving $(\textsc{CandidateChain}, C')$ do:

- if $||C'| - |C|| < k_c$ set $C := \mathsf{maxvalid}(C, C')$
- else set $i_c := i_c + k_c$ and send $C'[: i_c]$ to all parties in $\mathbb{V}$. Upon receiving $\lceil \frac{n}{2} \rceil$ messages $(\textsc{Checkpoint}, B||r)$ from different checkpointing parties, if $C'[i_c] = B||r$ set $C := C'$, else if $C'[i_c] = B$ set $C := C'[: i]||r$.

Upon receiving $(\textsc{Read})$ return $(\textsc{Chain}, C)$.

Prototype implementation

- PKI for checkpointing nodes
- 15 Amazon EC2 t2.micro nodes
- Raft: fail-stop consensus protocol
- $k_c = 4$
- Checkpoints are aggregated signatures
- Test blockchain: Private Ethereum Proof-of-Authority

Prototype evaluation

Storage (size of checkpoints):

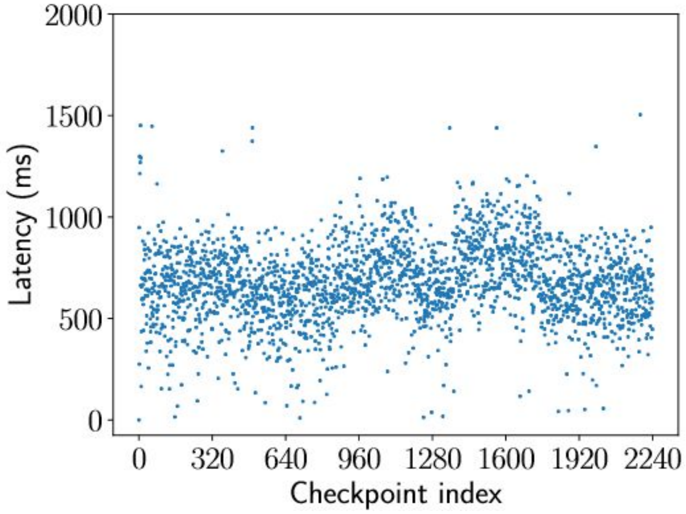
- 8 (nodes) · 64 (bytes of a single signature) = 512 bytes
- 0.6% increase in ledger's size

Prototype evaluation

Latency

(time between retrieval of block and issuing of signed checkpoint)

- London (EU): 557 ms
- N. California (US West): 620 ms
- São Paulo (South America): 711 ms
- Tokyo (Asia Pacific): 723 ms
- Singapore (Asia Pacific): 779 ms



Timestamps: Decentralized checkpoints

Functionality $\mathcal{F}_{\mathsf{Timestamp}}$

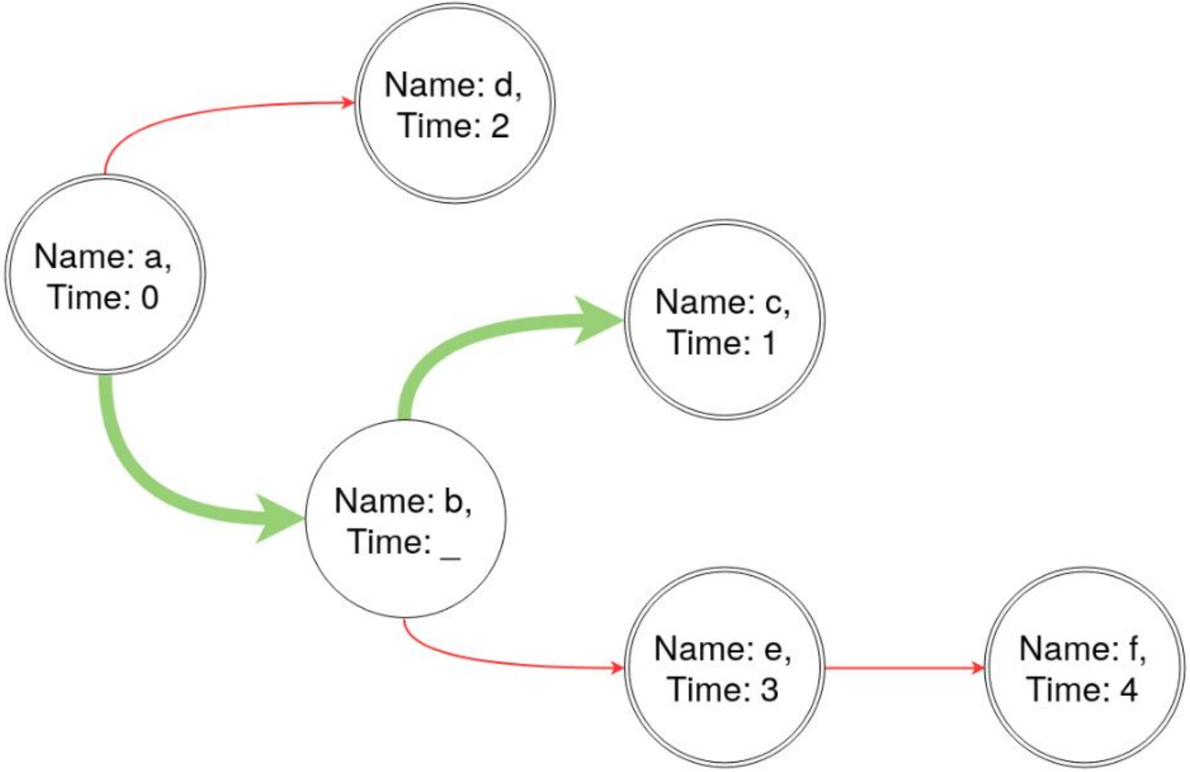
$\mathcal{F}_{\mathsf{Timestamp}}$ holds the following items:

- $T_{[]}$: an initially empty list of timestamped strings;
- τ: a counter initially set to 0;

Upon receiving $(\textsc{Timestamp}, s)$, if $\forall (s', \cdot) \in T_{[]} : s' \neq s$, set $\tau := \tau + 1$ and add (s, τ) to $T_{[]}$.

Upon receiving $(\textsc{Verify}, s, \tau)$, if $\exists (s, \tau) \in T_{[]}$ then return $(\textsc{VerifyTimestamp}, \top)$.

Chain decision using timestamps



Timestamping security

- Security guarantees: Same as checkpoints with kc = 1
- Timestamping *every block* is important:
  - A chain segment that follows a non-timestamped block can be removed in the future
- The entire block header needs to be timestamped:
  - Timestamping a hash is not enough, as the adversary can keep a timestamped block secret

Decentralized timestamping

Cost	Ethereum	Smart contract deployment	0.4$
		BTC* header timestamping	0.07$
		ETH* header timestamping	0.16$
	Bitcoin	BTC* header timestamping	0.45$
		ETH* header timestamping	3.6$
Latency	Ethereum	Stable timestamp	9 minutes
		Unstable timestamp	15 seconds
	Bitcoin	Stable timestamp	60 minutes
		Unstable timestamp	10 minutes
Proof size	Ethereum	Full node	181 GB
		SPV implementation	5 GB
		NIPoPoW implementation	6 MB
		FlyClient implementation	3 MB
	Bitcoin	Full node	240 GB
		SPV implementation	48 GB

Future work

- Byzantine Fault Tolerant checkpointing service
- Randomized checkpointing (intervals)
- Non-rushing adversaries
- Non-interactive (but centralised) timestamping
- Checkpoints for Proof-of-Stake

Conclusion

- In case of adversarial majority, an <u>external set of honest parties</u> needs to be introduced
- Checkpoints need to become <u>part of the chain</u> to ensure liveness
  - <u>Front-running attack</u>
- Checkpoints *can* be decentralized via <u>distributed ledger-based timestamping</u>

Thank you!