A Proof-of-Stake protocol for consensus on Bitcoin subchains

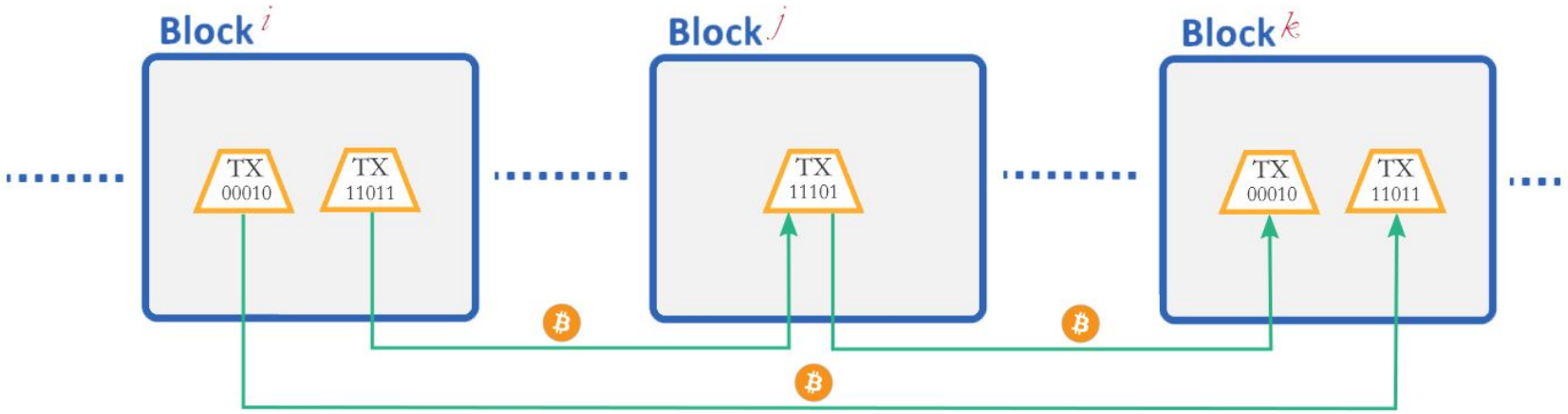
M. Bartoletti **Stefano Lande** A. S. Podda

University of Cagliari, Italy

Workshop on Trusted Smart Contracts, 2017

Bitcoin

Bitcoin is a popular cryptocurrency that uses a **blockchain** to store **transactions,** i.e. exchanges of **BTC** between client addresses



Less frequently, transactions are also used to embed a few bytes of **metadata**, usually via the **OP_RETURN** instruction

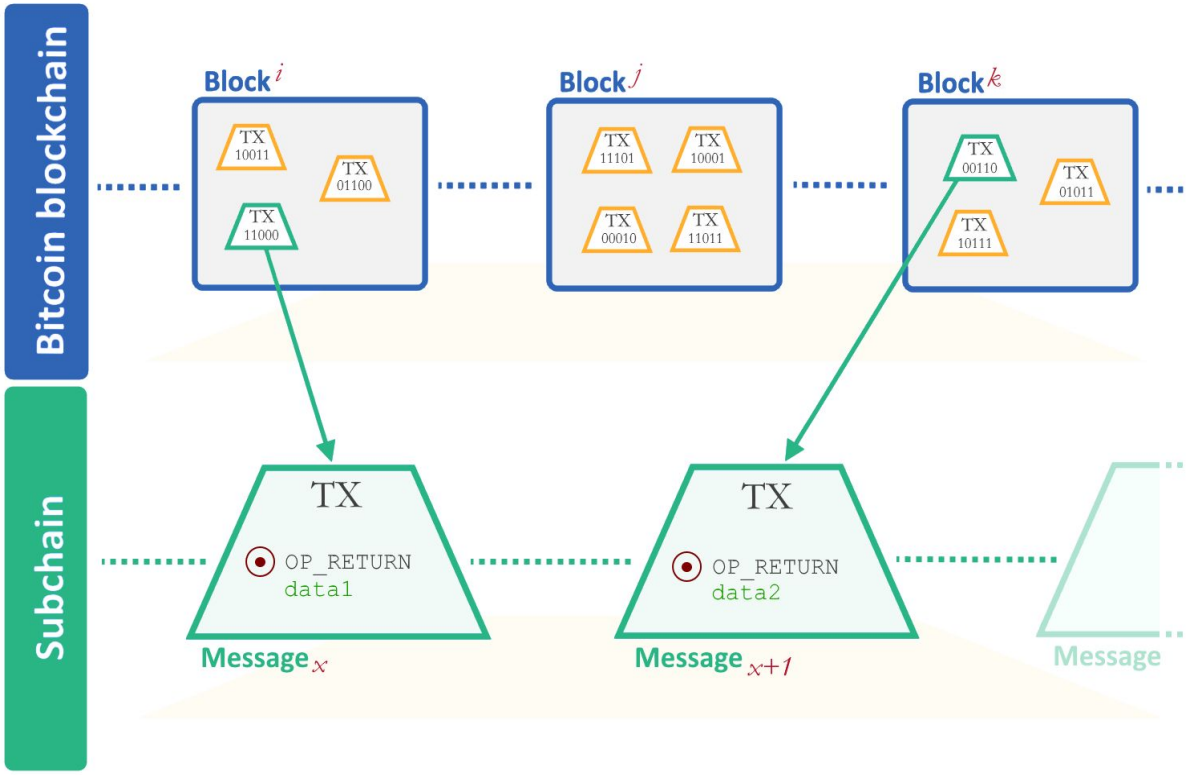
Subchains

Some platforms exploit metadata to store *tamper-proof* messages on the blockchain, examples:

EternityWall
(stores short
text messages)

Proof-Of-Existence
(stores hashes
of notary documents)

...



The sequence of messages of a platform forms a **subchain**

Subchains (2)

For EternityWall etc., there are no causal dependencies between messages, so you can rearrange them without losing the “consistency” of the subchain:

$$\sigma = \text{“Hello world”} :: \text{“P!=NP”} :: \text{“I’m happy”} \equiv \text{“I’m happy”} :: \text{“P!=NP”}$$

Vice versa, platforms that want to execute decentralized computations (eg: **smart contracts**) need to reach a consensus on the messages they publish

➔ less trivial to achieve “consistency”

Example

The smart contract **FACTORS_n**. Each client that extends the subchain with a new factor of **n** is rewarded by **1 BTC**. Two possible messages:

- (**A**, **x**) : the client **A** broadcast a new factor **x** of **n**
- (**pay**(1, **A**), **x**) : the client **A** is rewarded by **1 BTC** to have found **x**

Possible subchains for **FACTORS_330**:

σ_1 = (**A**, **11**) :: (**B**, **2**) :: (**pay**(1, **A**), **11**) :: (**pay**(1, **B**), **2**) ✅

σ_2 = (**A**, **11**) :: (**pay**(1, **A**), **11**) :: (**M**, **11**) ❌

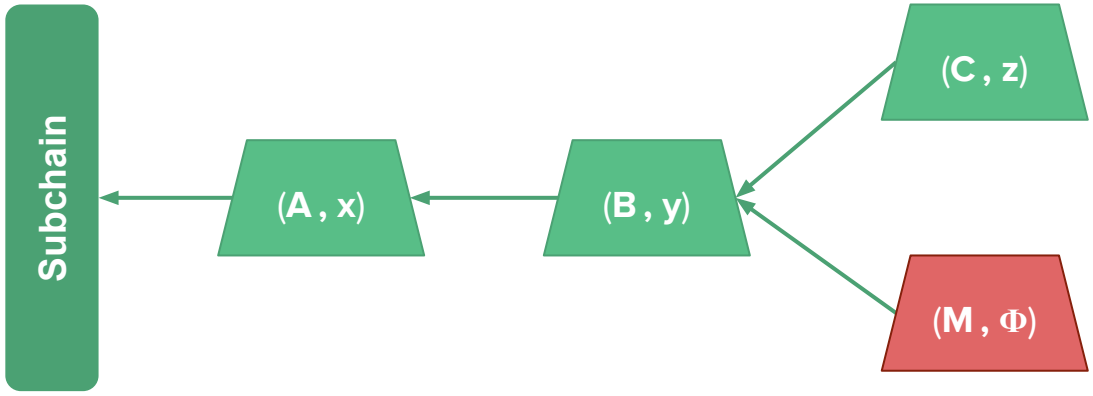
σ_3 = (**M**, **229**) :: (**pay**(1, **M**), **229**) ❌

σ_4 = (**A**, **11**) :: (**pay**(1, **M**), **11**) ❌

Consistent subchains

Bitcoin nodes cannot determine subchain consistency (they ignore metadata): they publish all messages indistinctly → **consensus between platform nodes** is required:

Which is the next valid subchain message?



Well-known existing platforms (eg: Counterparty) do not use a consensus mechanism. As consequence, each node has its own view of the subchain

Contribution

We propose **a protocol** that allows platform nodes to reach consensus on subchains built upon the Bitcoin blockchain:

- by specifying how platform nodes must uniquely choose the next update;
- by economically penalizing dishonest nodes (i.e., those violating the protocol)

The protocol implements a **Proof-of-Stake**

Proof-of-Stake upon Proof-of-Works *vs.* pure Proof-of-Stake

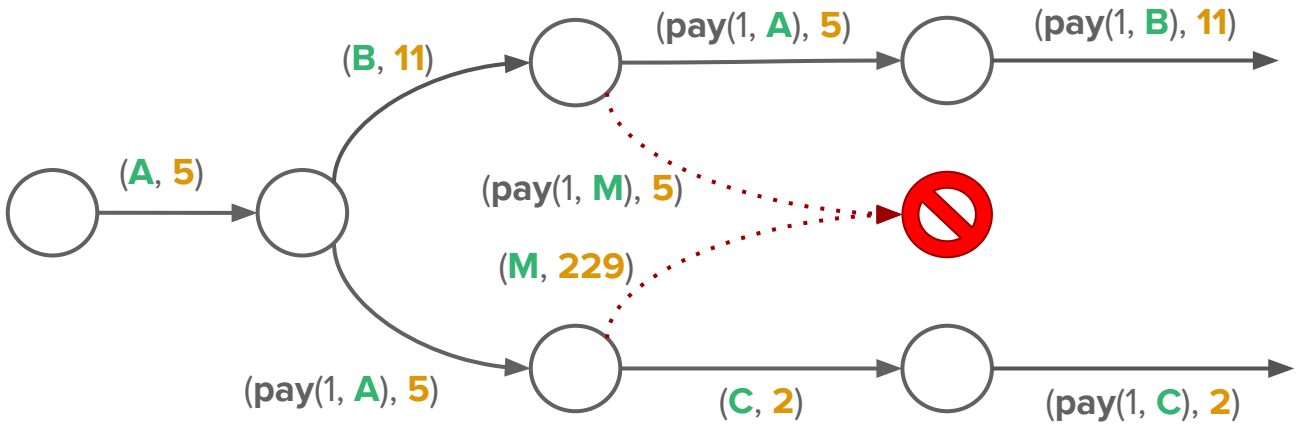
The model

We abstractly model platform-specific computations as an **LTS**

A **label** of the LTS has the form (**A**, **x**), denoting that client **A** publishes the update message **x**

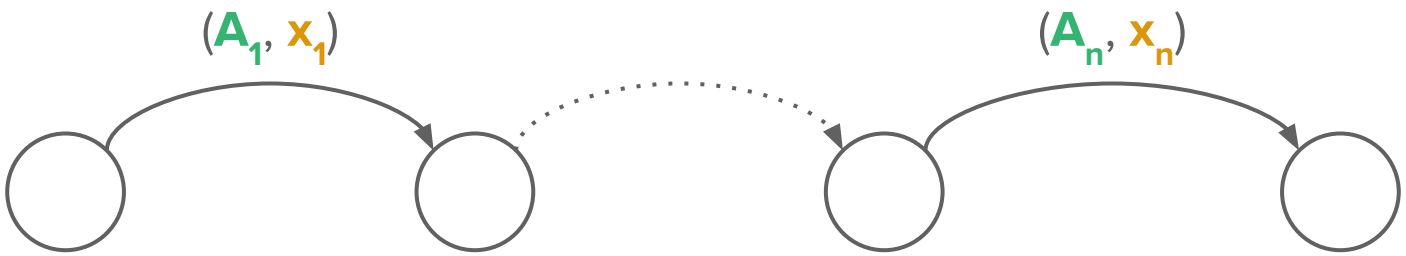
We use a special label (**A, pay**(v, **B**)) to indicate an update message that also transfers v BTC from **A** to **B**

Example: a finite fragment of the LTS of **FACTOR_330**



Consistency

We say that a subchain λ = (A_1, x_1) … (A_n, x_n) is **consistent** whenever λ is a path of the LTS

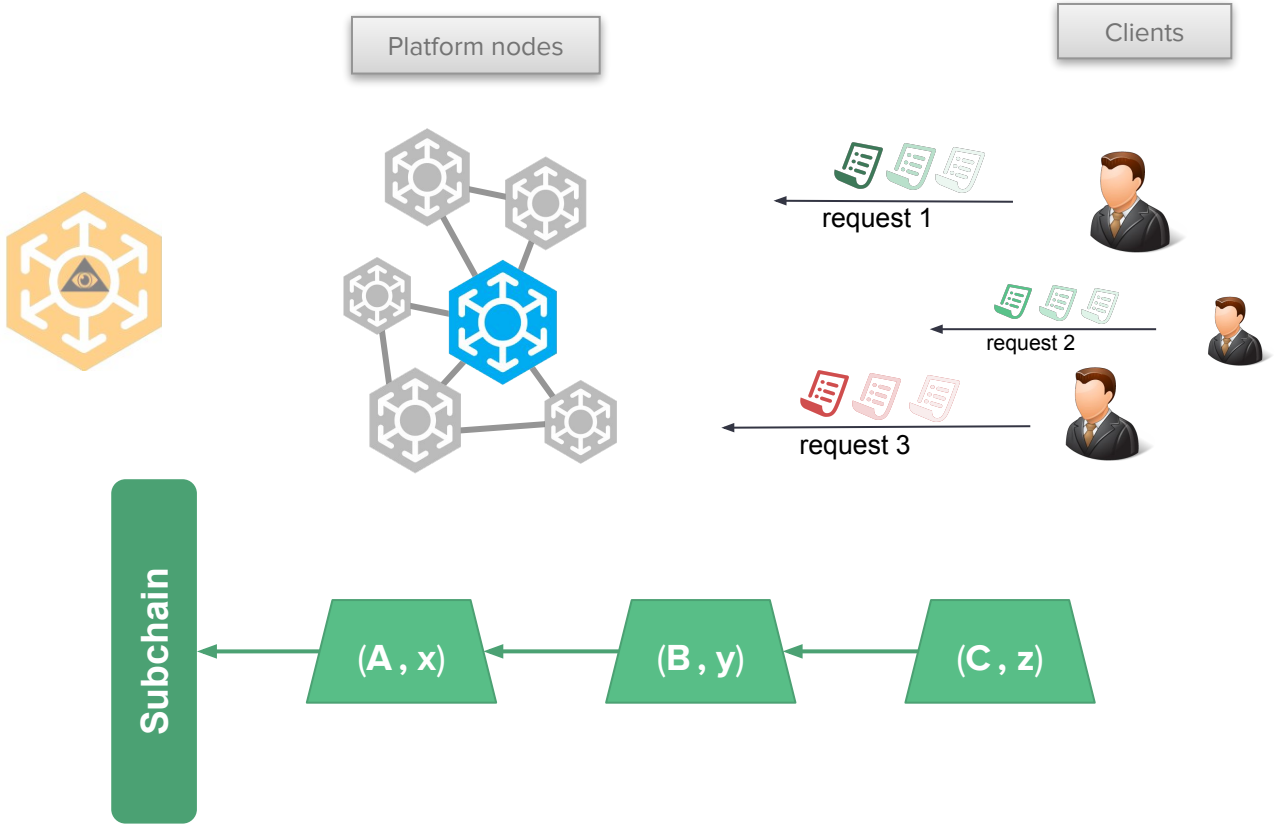


An update (**A**, **x**) is consistent when the new subchain, obtained appending it to the current suchain, is consistent

Protocol

The protocol is organized in stages of fixed duration

At the begin of each stage, clients send their update requests to the network of platform nodes...

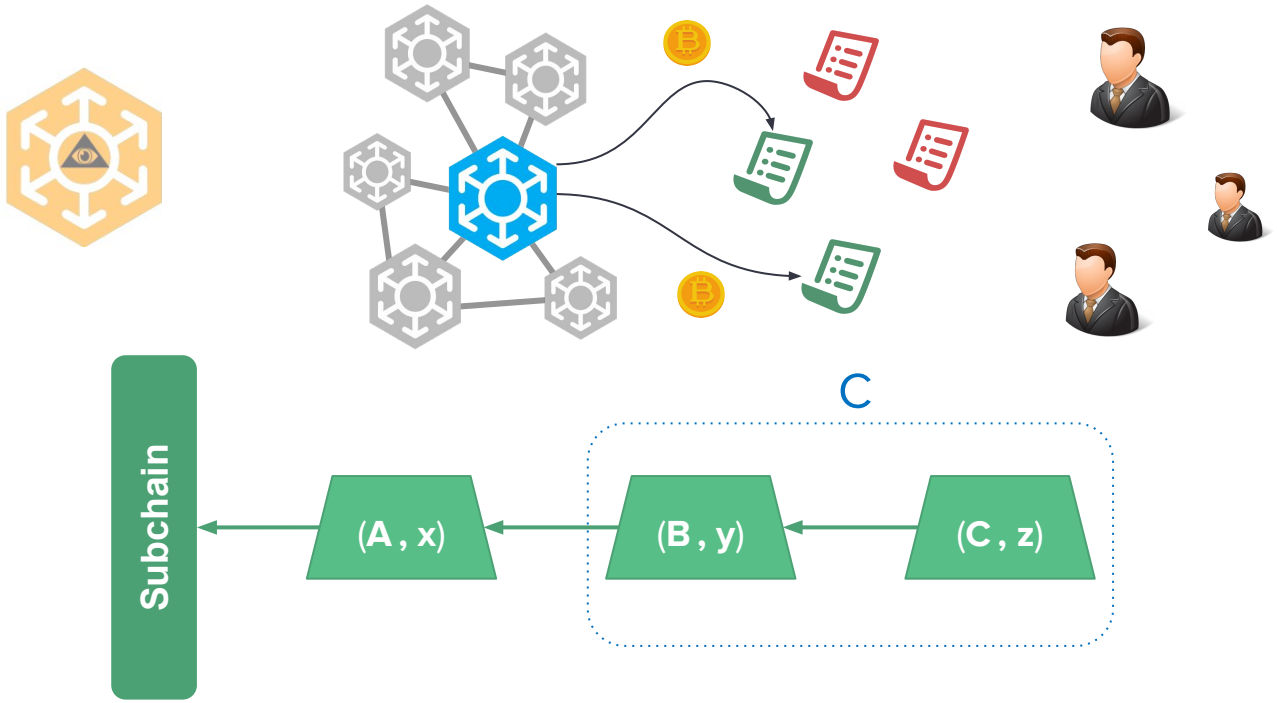


Protocol (2)

... then, each platform node votes the updates that it considers consistent

To vote a request, a node must:

- invest K **BTC** on it ⇒ K is a fixed **stake** amount
- confirm a message previously published on the subchain

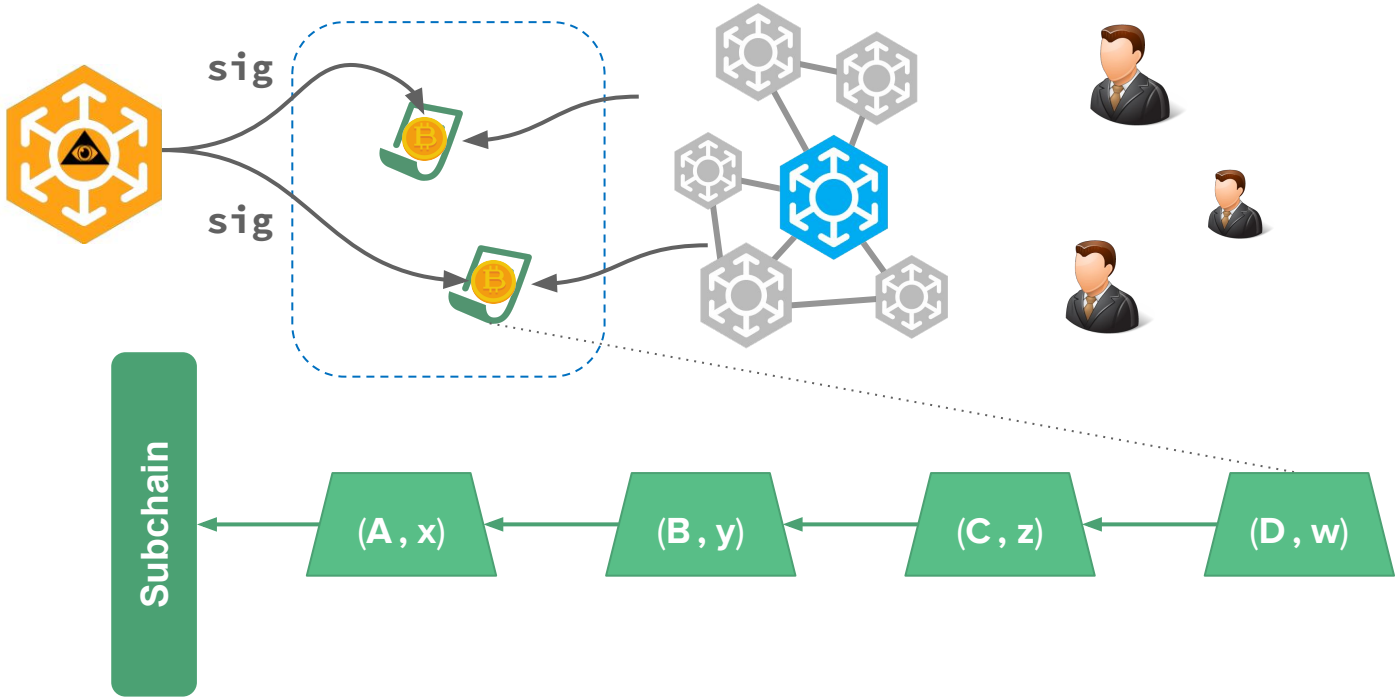


Protocol (3)

... then, nodes send voted request to the **request pool**

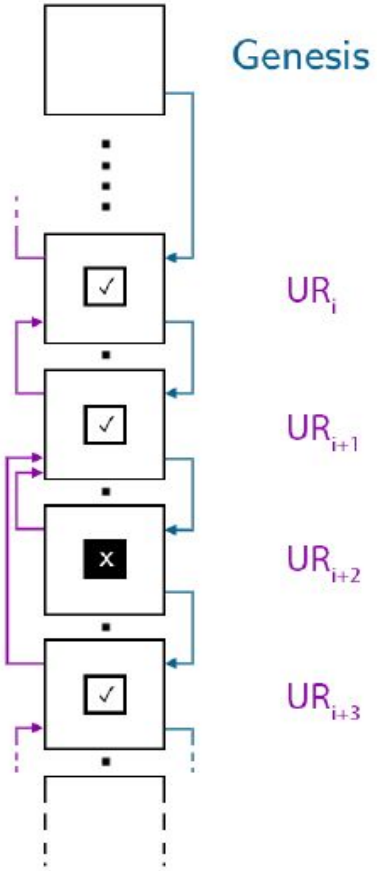
The **arbiter** sign all well-formed request

The nodes send all signed request to Bitcoin node. Only one transaction will appear in the new block



Implementation on Bitcoin

UR_i
in[0]: $Fee_i[out]$ in-script[0]: $sig_C(\bullet)$
in[1]: $Stake_i[out]$ in-script[0]: $sig_N(\bullet)$
in[2]: $Confirm_{i-1}[out_1]$ in-script[0]: $sig_T(\bullet)$
out-script[0](): OP_RETURN A : a value[0]: 0
out-script[1](T, σ): $ver_T(T, \sigma)$ value[1]: 0.0001
out-script[2](T, σ): $ver_{N'}(T, \sigma)$ value[2]: κ + fee
out-script[3](T, σ): $ver_B(T, \sigma)$ value[3]: v_{pay}
lockTime: $n + 1$



Standard transactions ✓

Properties of the protocol

Let S be the total stake of the network, and S_h the total stake of honest platform nodes

In a given protocol stage:

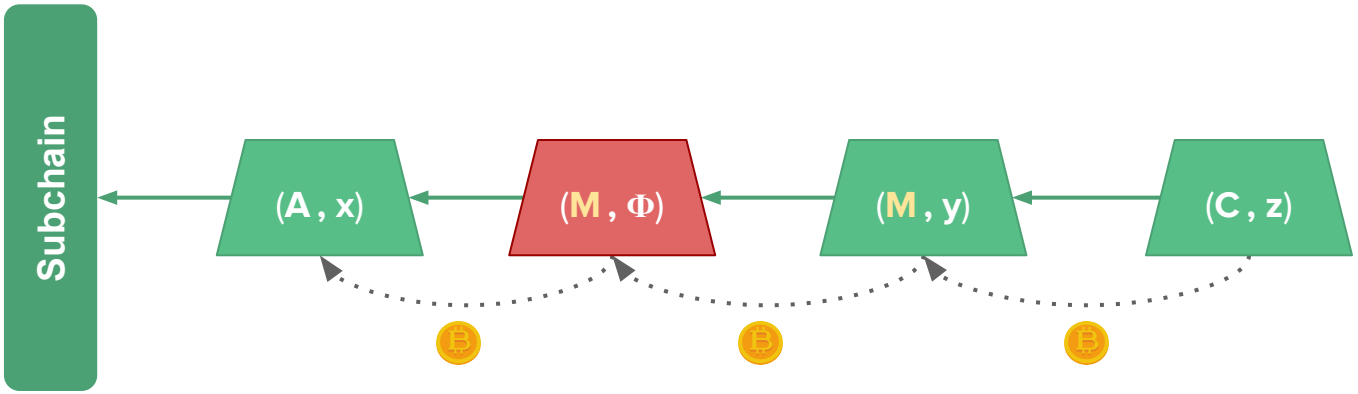
- the probability that an honest node (with stake h) updates the subchain is **at least** h/S

- the probability that a dishonest node updates the subchain is **at most** $(S - S_h)/S$

Self-compensation attack

The attacker can publish an inconsistent update, the appends a consistent one to get its first stake back

An honest node will confirm the second update, so the attacker append an inconsistent update without losing its stake



Self compensation attack (2)

The probability *p* of an attacker succeed in a self-compensation attack is at most:

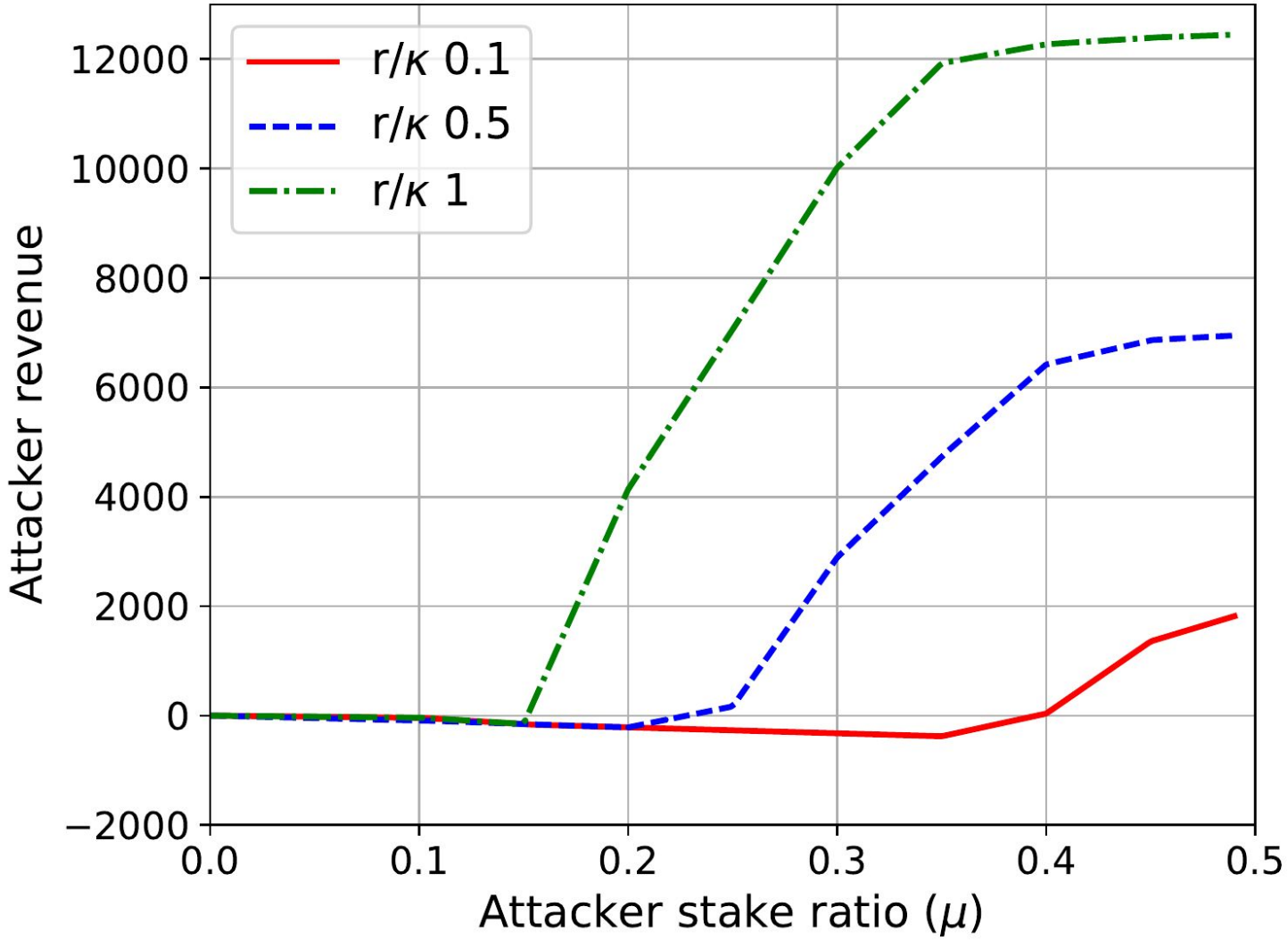
$$\binom{C}{2} \cdot \mu^2 (1-\mu)^{C-2}$$

Where C is the checkpoint offset, μ is the attacker stake over the total

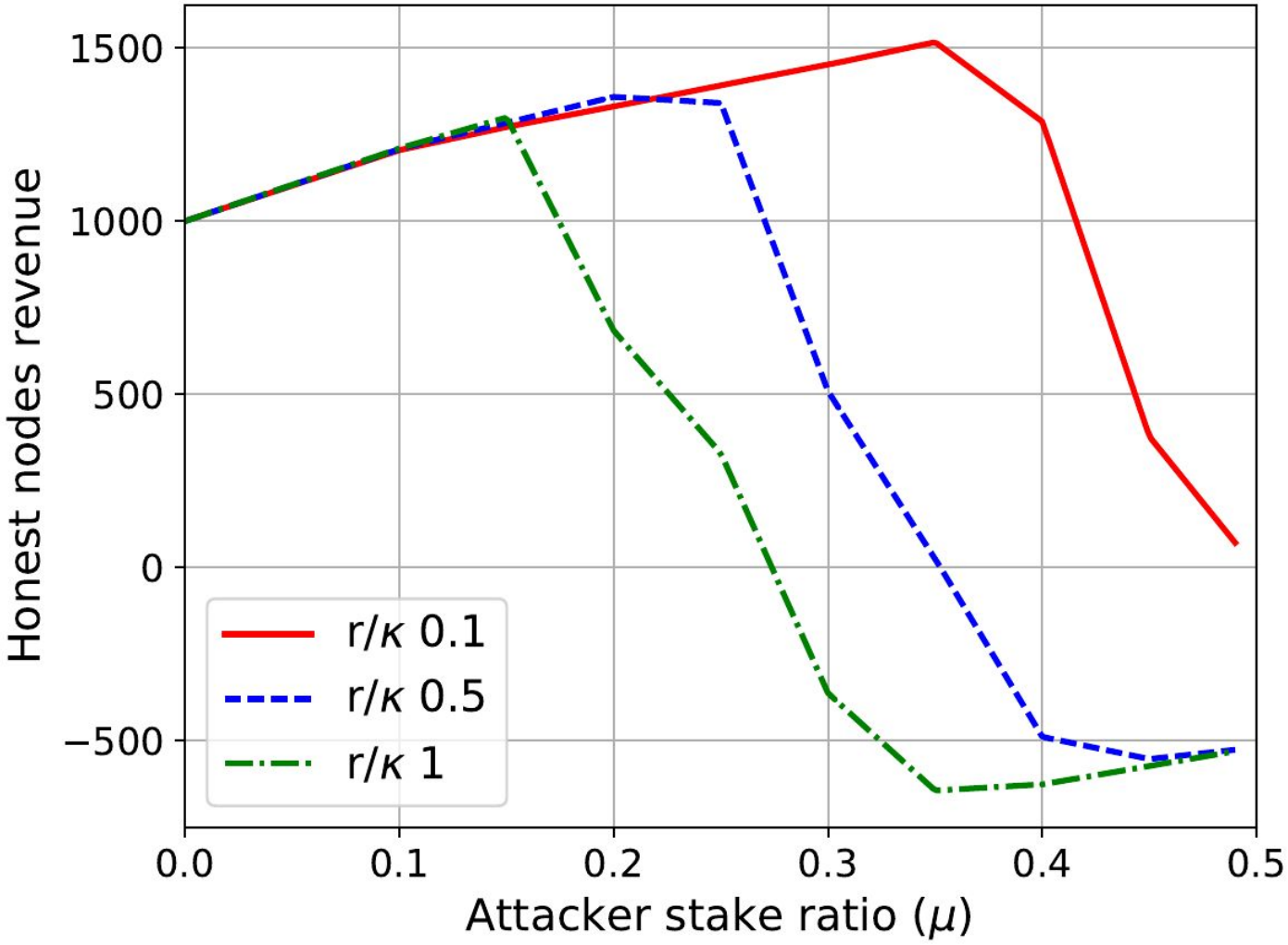
The probability grows with C. For instance, for μ = 0.01:

- C = 2 ➔ *p* = 0.0001
- C = 3 ➔ *p* = 0.000297
- C = 4 ➔ *p* = 0.00058806

Experimental validation



Experimental validation (2)



Conclusions

- Proof-of-Stake over Bitcoin
  - Allow to maintain **consistent subchains**
  - Economic disincentive to dishonest platform nodes

- Future works:
  - Develop a programming language for smart contracts
  - Implement a framework to publish and execute smart contracts

Thank you!