Brick: Asynchronous Incentive-Compatible Payment Channels

Zeta Avarikioti[1], Eleftherios Kokoris-Kogias[2], Roger Wattenhofer[1], and Dionysis Zindros[3]

[1] ETH Zürich
[2] IST Austria, Novi Research
[3] NKUA, IOHK

Abstract. Off-chain protocols (channels) are a promising solution to the scalability and privacy challenges of blockchain payments. Current proposals, however, require synchrony assumptions to preserve the safety of a channel, leaking to an adversary the exact amount of time needed to control the network for a successful attack. In this paper, we introduce Brick, the first payment channel that remains secure under network asynchrony and concurrently provides correct incentives. The core idea is to incorporate the conflict resolution process within the channel by introducing a rational committee of external parties, called wardens. Hence, if a party wants to close a channel unilaterally, it can only get the committee's approval for the last valid state.
Additionally, Brick provides sub-second latency because it does not employ heavy-weight consensus. Instead, Brick uses consistent broadcast to announce updates and close the channel, a light-weight abstraction that is powerful enough to preserve safety and liveness to any rational parties. We formally define and prove for Brick the properties a payment channel construction should fulfill. We also design incentives for Brick such that honest and rational behavior aligns. Finally, we provide a reference implementation of the smart contracts in Solidity.

1 Introduction

The prime solution to the scalability challenge [12] of large-scale blockchains, are the so-called *channels* [13, 35, 38]. The idea is that any two parties that interact (often) with each other can set up a joint account on the blockchain, i.e., a channel. Using this channel, the two parties can transact off-chain, sending money back and forth by just sending each other signed messages. The two parties are relying on the blockchain as a fail-safe mechanism in case of disputes.

The security guarantees of a channel are ensured by a dispute handling mechanism. If one party tries to cheat the other party, in particular by trying to close a channel on the underlying blockchain in an invalid (outdated) state, then the attacked party has a window of time (t) to challenge the fraud attempt. Hence, a channel is secure as long as all parties of the channel are frequently – at least once in t time – online and monitoring the blockchain. This is problematic in real networks [31], as one party may simply execute a denial-of-service (DoS) attack

on the other party. To add insult to injury, the dispute period t is public; the attacking party hence knows the exact duration of the denial-of-service attack.

The issue is well-known in the community, and there were solution attempts using semi-trusted third parties called *watchtowers* [2, 5, 7, 16, 30]. The idea is that worrisome channel parties can hire watchtowers that watch the blockchain on their behalf in case they were being attacked. So instead of DoSing a single machine of the channel partner, the attacker might need to DoS the channel partner as well as its watchtower(s). This certainly needs more effort as the adversary must detect a watchtower reacting and then block the dispute from appearing on-chain. However, if large amounts of money are in a channel, it will easily be worth the investment.

While DoS attacks are also possible in blockchains such as the Bitcoin blockchain, DoS attacks on channels have a substantially different threat level. A DoS attack on a blockchain is merely a liveness attack: One may prevent a transaction from entering the blockchain at the time of the attack. However, the parties involved with the transaction will notice this, and can simply re-issue their transaction later. A DoS attack on a channel, on the other hand, will steal all the funds that were in the channel. Once the fraudulent transaction is in the blockchain, uncontested for t time, the attack succeeds, and nobody but the cheated party (and its watchtowers) will know any better.

Channels need a more fundamental solution. Not unlike blockchains, introducing timing parameters is acceptable for liveness. Security on the other hand should be guaranteed even if the network behaves completely asynchronously. To that end we introduce Brick, a novel incentive-compatible payment channel construction that does not rely on timing assumptions for the delivery of messages to be secure. Brick provides *proactive security*, detecting and preventing fraud before it appears on-chain. As a result, Brick can guarantee the channels' security even under censorship[4] [31] or any liveness attack.

To achieve these properties, Brick needs to address three key challenges. The first challenge is how to achieve this proactive check without using a single trusted third party that approves every transaction [41]. The core idea of Brick is to provide proactive security to the channel instead of reactive dispute resolution. To this end, Brick employs a *group of wardens*. If there is a dispute, the wardens make sure the correct state is the only one available for submission on-chain, regardless of the amount of time it takes to make this final state visible. The second challenge for Brick is cost. To simulate this trusted third party, it would need the wardens to run costly asynchronous consensus [26] for every update. Instead, in Brick we show that a light-weight *consistent broadcast* protocol is enough to preserve both safety and liveness.

A final challenge of Brick that we address is *incentives*. While the wardens may be partially byzantine, we additionally want honest behavior to be their dominant strategy. Unfortunately, existing watchtower solutions do not align the expected and rational behavior of the watchtower, hence a watchtower is

[4] This censoring ability is encompassed by the chain-quality property [19] of blockchain systems which is rightly bound to the synchrony of the network.

reduced to a trusted third party. Specifically, Monitors [16], Watchtowers [2], and DCWC [5] pay the watchtower upon fraud. Given that the use of a watchtower is public knowledge, any rational channel party will not commit fraud and hence the watchtower will never be paid. Therefore, there is no actual incentive for a third party to offer a watchtower service. On the other hand, Pisa [30] pays the watchtower regularly every time a transaction is executed on the channel. The watchtower also locks collateral on the blockchain in case it misbehaves. However, Pisa's collateral is not linked to the channel or the party that employed the watchtower. Hence, a watchtower that is contracted by more than one channel can double-assign the collateral, making Pisa vulnerable to bribing attacks. Even if the incentives of Pisa get fixed, punishing a misbehaving watchtower in Pisa is still a synchronous protocol (though for a longer period). In Brick, we employ both rewards and punishment to design the appropriate incentives such that honest and rational behavior of wardens align, while no synchrony assumptions are required, i.e., the punishment of misbehaving wardens is not conditional on timing assumptions.

To evaluate our channel construction we deploy our protocol on a large-scale testbed and show that the overhead of an update is around the round-trip latency of the network (in our case 0.1 seconds). Unlike existing channels, the parties in Brick need not wait for the dispute transaction to appear on-chain. Hence, our dispute resolution mechanism is three orders of magnitude faster than existing blockchain systems that need to wait until the transaction is finalized on-chain. We additionally implement the on-chain operations of Brick in a Solidity smart contract that can be deployed on the Ethereum blockchain. We provide gas measurements for typical operations on the smart contract illustrating that it is practical. Our smart contract implementation is well tested and can be adopted towards a real deployment of Brick.

In summary, this paper makes the following contributions:

- We introduce Brick, the first incentive-compatible off-chain construction that operates securely with offline channel participants under full asynchrony with sub-second latency.
- We define the desired channel properties and show they hold for Brick under a hybrid model of rational and byzantine participants (channel parties and wardens). Specifically, we present elaborate incentive mechanisms (rewards and punishments) for the wardens to maintain the channel properties under collusion or bribing.
- We evaluate the practicality of Brick by fully implementing its on-chain functionality in Solidity for the Ethereum blockchain. We measure its operational costs in terms of gas and illustrate that its deployment is practical.

2 Protocol Overview

2.1 System Model

Cryptographic assumptions. We make the usual cryptographic assumptions: the participants are computationally bounded and cryptographically-secure communication channels, hash functions, signatures, and encryption schemes exist.

Blockchain assumptions and network model. We assume that any message sent by an honest party will be delivered to any other honest party within a polynomial number of rounds. We do not make any additional assumptions about the network (e.g., known bounds for message delivery). Furthermore, we do not require a "perfect" blockchain system since Brick can tolerate temporary liveness attacks. Specifically, if an adversary temporarily violates the liveness property of the underlying blockchain, this may result in violating the liveness property of channels but will not affect the safety. Nevertheless, we assume the underlying blockchain satisfies persistence [19]. In Section 7, we discuss a modification of Brick that is safe *even when persistence is temporarily violated.*

Threat model. We initially assume that at least one party in the channel is honest to simplify the security analysis. However, later, we show that the security analysis holds as long as the "richest" party of the channel is rational and intentionally deviates from the protocol only if it can increase its profit (utility function). Regarding the committee, we assume that there are at most f out of $n = 3f + 1$ byzantine wardens, and we define a threshold $t = 2f + 1$ to achieve the liveness and safety properties. The non-byzantine part of the committee is assumed rational; we first prove the protocol goals for t honest wardens, and subsequently align the rational behavior to this through incentives.

2.2 Brick Overview

Both parties of a channel agree on a committee of wardens before opening the channel. The wardens commit their identities on the blockchain during the funding transaction of the channel (opening of the channel). After opening the channel on the blockchain, the channel can only be closed either by a transaction published on the blockchain and signed by both parties or by a transaction signed by one of the parties and a threshold (t) of honest wardens. Thus, the committee acts as power of attorney for the parties of the channel. Furthermore, Brick employs correct incentives for the t rational wardens to follow the protocol, hence it can withstand $t = 2f + 1$ rational and f byzantine wardens, while the richest channel party is assumed rational and the other byzantine.

A naive solution would then instruct the committee to run asynchronous consensus on every new update, which would cost $O(n^4)$ [26] per transaction, a rather big overhead for the critical path of our protocol. Instead in Brick, consensus is not necessary for update transactions, as we only provide guarantees to rational parties (if both parties misbehave one of them might lose its funds). As a result, every time a new update state occurs in the channel (i.e., a transaction), the parties run a consistent broadcast protocol (cost of $O(n)$) with the committee. Specifically, a party announces to each warden that a state update has occurred. This announcement is a monotonically increasing sequence number to guarantee that the new state is the freshest state, signed by both parties of the channel to signal that they are in agreement. If the consistent broadcast protocol succeeds (t wardens acknowledge reception) then this can serve as proof for both parties that the state update is safe. After this procedure terminates correctly, both parties proceed to the execution of the off-chain state.

At the end of the life-cycle of a channel, a dispute might occur, leading to the unilateral closing of the channel. Even in this case, we can still guarantee the security and liveness of the closing with consistent broadcast. The crux of the idea is that if $2f+1$ wardens accepted the last sequence number before receiving the closing request (hence the counterparty has committed), then at least one honest warden will not accept the closing at the old sequence number. Instead, the warden will reply to the party that it can only close at the state represented by the last sequence number. As a result we define a successful closing to be at the maximum of all proposed states, which guarantees safety. *Although counter-intuitive, this closing process is safe because the transactions are already totally ordered and agreed to by the parties of the channel;* thus, the committee simply acts as shared memory of the last sequence number.

2.3 Reward Allocation & Collateral

To avoid bribing attacks, we enforce the wardens to lock collateral in the channel. The total amount of collateral is proportional to the value of the channel meaning that if the committee size is large, then the collateral per warden is small. More details on the necessary amount of collateral are thoroughly discussed in Sections 3.2 and 7. Additionally, the committee is incentivized to actively participate in the channel with a small reward that each warden gets when they acknowledge a state update of the channel. This reward is given with a unidirectional channel [22], which does not suffer from the problems BRICK solves. Moreover, the wardens that participate in the closing state of the channel get an additional reward, hence the wardens are incentivized to assist a party when closing in collaboration with the committee is necessary.

2.4 Protocol Goals

To define the goals of BRICK, we first need to define the necessary properties of a channel construction. Intuitively, a channel should ensure similar properties with a blockchain system, i.e., a party cannot cheat another party out of its funds, and any party has the prerogative to eventually spend its funds at any point in time. The first property, when applied to channels, means that no party can cheat the channel funds of the counterparty, and is encapsulated by *Safety*. The second property for a channel solution is captured by *Liveness*; it translates to any party having the right to eventually close the channel at any point in time. We say that a channel is *closed* when the locked funds of the channel are spent on-chain, while a channel *update* refers to the off-chain change of the channel's state. In addition, we define *Privacy* which is not guaranteed in many popular blockchains, such as Bitcoin [33] or Ethereum [40], but constitutes an important practical concern for any functional monetary (cryptocurrency) system.

First, we define some characterizations on the state of the channel, namely, validity and commitment. Then, we define the properties for the channel construction. Each state of the channel has a discrete sequence number that reflects the order of the state. We assume the initial state of the channel has sequence number 1 and every new state has a sequence number one higher than the pre-

vious state agreed by both parties. We denote by s_i the state with sequence number i.

Definition 1. *A state of the channel, s_i, is **valid** if the following hold:*
- *Both parties of the channel have signed the state s_i.*
- *The state s_i is the **freshest** state, i.e., no subsequent state s_{i+1} is valid.*
- *The committee has not invalidated the state. The committee can invalidate the state s_i if the channel closes in the state s_{i-1}.*

Definition 2. *A state of the channel is **committed** if it was signed by at least $2f + 1$ wardens or is valid and part of a block in the persistent[5] part of the blockchain.*

Definition 3 (Safety). *A* Brick *channel will only close in the freshest committed state.*

Definition 4 (Liveness). *Any valid operation (update, close) on the state of the channel will eventually[6] be committed (or invalidated).*

Definition 5 (Privacy). *No external (to the channel) party learns about the state of the channel (e.g., the current distribution of funds between the parties of a payment channel) unless at least one of the parties initiate the closing of the channel.*

3 Brick Design

In this section, we first present the Brick architecture assuming t honest wardens, and then introduce the incentive mechanisms aligning honest and rational behavior.

3.1 Architecture

Brick consists of three phases: *Open*, *Update*, and *Close*. We assume the existence of a smart contract that has two functions, Open and Close, which receive the inputs of the protocols and verify that they adhere to the abstractly defined protocols specified below.

Protocol 1 describes the first phase, *Open*, which is the opening of a channel between two parties. In this phase, the parties create the initial funding transaction, similarly to other known payment channels such as [13, 35]. However, in Brick we also define two additional parameters in the funding transaction: the hashes of the public keys of the wardens of the channel, denoted by $W_1, W_2, \ldots, W_n$, and the threshold t.

The second phase, *Update*, consists of two protocols, Protocol 2 (Update), and Protocol 3 (Consistent Broadcast). Both algorithms are executed consecutively every time an update occurs, i.e., when the state of the channel changes. In Protocol 2, the parties of the channel agree on a new state and create an

[5] The part of the chain where the probability of fork is negligible hence there is transaction finality, e.g., 6 blocks in Bitcoin.

[6] Depending on the message delivery.

Protocol 1: BRICK Open

Data: Parties A, B, wardens $W_1, \ldots, W_n$, initial state s_1.
Result: Open a BRICK payment channel.

/* The parties agree on the first update before opening the channel */
1. Register to $\{M, \sigma(M)\}$ the announcement of Protocol 2 on input (A, B, s_1).

/* The parties broadcast the first sequence number to the wardens */
2. Execute Protocol 3 on input $(M, \sigma(M), A, B, W_1, W_2, \ldots, W_n)$.
// without an update fee

/* The parties open the BRICK channel */
3. Both parties A, B sign and publish on-chain
$open(H(W_1), H(W_2), \ldots, H(W_n), t, s_1)$.
/* A closing fee F is included in the funding transaction, as well as the collateral C of each warden along with their signature. */

announcement, which they subsequently broadcast to the committee with Protocol 3. To agree on a new state, both parties sign the hash of the new state[7]. This way both parties commit to the new state of the channel, while none of the parties can unilaterally close the channel without the collaboration of either the counterparty or the committee. The announcement, on the other hand, is the new sequence number signed by both parties of the channel[8]. The signed sequence number allows the wardens to verify agreement has been reached between the channel parties on the new state, while the state of the channel remains private. Upon receiving a valid announcement from a party, wardens reply with their signature on the announcement. A party executes the new state update when it receives t signatures from the wardens.

The last phase of the protocol, *Close*, can be implemented in two different ways: the first is similar to the traditional approach for closing a channel (Protocol 4: Optimistic Close) where both parties collectively sign the freshest state (closing transaction) and publish it on-chain. However, in case a channel party is not responding to new state updates or closing requests, the counterparty can unilaterally close the channel in collaboration with the committee of the channel (Protocol 5: Pessimistic Close).

In Protocol 5, a party requests from each warden its signature on the last committed sequence number. A warden, upon receiving the closing request, publishes on-chain a closing announcement, i.e., the stored sequence number signed

[7] Blinding the commitment to the state is not necessary for BRICK, but we do it for compatibility with an auditable extension of BRICK [6] where the hash of the state is given to the wardens along with the sequence number. Because the states of a channel may be limited, the salt r_i is used to prevent wardens from retrieving the state by simply hashing all possible states, effectively compromising privacy.

[8] We abuse the notation of signature σ to refer to the multisig of both A and B.

Protocol 2: BRICK Update

Data: Parties A, B, current state s.
Result: Create announcement $M, \sigma(M)$ (sequence number of new state signed by both parties).

1. Both parties A, B sign, exchange, and store: $\{H(s_i, r_i), i\}$, where r_i is a random number and s_i the current state. `// The parties store only the current and previous hash`

2. Upon receiving the signature of the counterparty on $\{H(s_i, r_i), i\}$, a party replies with its signature on the sequence number $\sigma(i)$. `// creating the announcement` $\{M, \sigma(M)\}(M = i)$

Protocol 3: BRICK Consistent Broadcast

Data: Parties A, B, wardens $W_1, \ldots, W_n$, announcement $\{M, \sigma(M)\}$.
Result: Inform the committee of the new update state and verify the validity of the new state.

1. Each party broadcasts to all the wardens $W_1, W_2, \ldots, W_n$ the announcement $\{M, \sigma(M)\}$. `// alongside a fee` r

2. Each warden W_j, upon receiving $\{M, \sigma(M)\}$, verifies that both parties' signatures are present, and the sequence number is exactly one higher than the previously stored sequence number. If the warden has published a closing state, it ignores the state update. Otherwise, W_j stores the announcement $\{M, \sigma(M)\}$ (replacing the previous announcement), signs M, and sends the signature $\sigma_{W_j}(M)$ to the parties. `// only to the parties that payed the fee`

3. Each party, upon receiving at least t signatures on the announcement M, considers the state committed and proceeds to the state transition.

along with a flag close. When t closing announcements are on the persistent part of the chain, the party recovers the state that corresponds to the maximum sequence number from the closing announcements s_i. Then, the party publishes state s_i and the random number r_i along with the signatures of both parties on the corresponding hash and sequence number $\sigma(H(s_i, r_i), i)$ on-chain. As soon as these data are included in a (permanent) block, the BRICK smart contract performs the following operations: (a) recovers from the submitted state s_i and salt r_i the hash $H(s_i, r_i)$ and the maximum sequence number i, (b) verifies that the signatures of both parties are on the message $\{H(s_i, r_i), i\}$, and (c) there are t submitted announcements that correspond to warden identities committed on-chain in Protocol 1. If all verifications check the smart contract closes the channel in the submitted state s_i.

3.2 Incentivizing Honest Behavior

BRICK actually works without the fees, if we assume one honest party and t honest wardens. However, our goal is to have no honest assumptions and instead

Protocol 4: BRICK Optimistic Close

Data: Parties A, B, state s.
Result: Close a channel on state s, assuming both parties are responsive and in agreement.

1. A party $p \in \{A, B\}$ broadcasts the request $close(s)$.

2. Both parties A, B sign the state s (if they agree) and exchange their signatures.

3. The party p (or any other channel party) publishes the signed by both parties state, $\sigma_{A,B}(s)$ on-chain.

/* The collateral C is returned to each warden */
/* The closing fee F is returned to the parties */

Protocol 5: BRICK Pessimistic Close

Data: Party $p \in \{A, B\}$, wardens $W_1, \ldots, W_n$, state s_i, random nonce r_i.
Result: Close a channel on state s_i with the assist of the committee.

1. Party p broadcasts to the wardens $W_1, W_2, \ldots, W_n$ the request $close()$.

2. Each warden W_j publishes on-chain a signature on the (last) stored announcement $\sigma_{W_j}(M, close)$ and stops signing new state updates.

3. Party p, upon verifying t on-chain signed announcements by the wardens, recovers the $max(i)$ that is included in the announcements. Then, party p publishes on-chain the state s_i, the random number r_i, and the signature of both parties on $\{H(s_i, r_i), i\}$.

4. After the state is included in a (permanent) block, the smart contract recovers $\{H(s_i, r_i), i\}$, verifies both parties' signatures and the wardens identities, and then closes the channel in state s_i.

align rational behavior to honest through incentives. There are three incentive mechanisms in BRICK:

Update Fee (r). The parties establish a unidirectional channel [22] with each warden and send a fee when they want a signature for a state update. Note that the update fee is awarded to the wardens at step 1 of Protocol 3.

Closing Fee (F). During phase *Open* (Protocol 1), the parties lock a closing fee F in the channel. If a party closes in a collaboration with the wardens, the closing fee is split only among the first t wardens that publish an announcement on-chain (see Protocol 6). If the channel closes optimistically (Protocol 4), the closing fee returns to the parties.

Collateral (C). During phase *Open*, each warden locks collateral C at least equal to the amount locked in the channel v divided by f. If a warden misbehaves, the closing party can claim the warden's collateral by submitting a proof-of-fraud in the BRICK smart contract during phase *Close*; otherwise, the collateral

is returned to the warden when the channel closes (Protocol 6). A proof-of-fraud consists of two conflicting messages signed by the same warden: (a) a signature on an announcement on a state update of the channel, and (b) a signature on an announcement for closing on a previous state of the channel.

In case, a party submits $x \leq f$ proofs-of-fraud, the closing process is extended until $x+t$ wardens have published an announcement on-chain. Then, the channel closes in the state with the maximum sequence number from the announcements submitted by the t non-cheating wardens. On the other hand, if a party submits at least $f+1$ proofs-of-fraud, the party that submitted the proofs-of-fraud claims only the collateral from the cheating wardens, while the entire channel balance is awarded to the counterparty. If no proofs-of-fraud are submitted the channel closes as described in Protocol 5, as it is a subcase of Protocol 6 for $x = 0$.

Protocol 6: BRICK Pessimistic Close with Incentives

Data: Party $p \in \{A, B\}$, wardens $W_1, \ldots, W_n$, state s_i, random nonce r_i.
Result: Close a channel on state s_i with the assist of the committee.

`/* Similarly to Protocol 5 */`

1. Party p broadcasts to the wardens $W_1, W_2, \ldots, W_n$ the request $close()$.

2. Each warden W_j publishes on-chain a signature on the (last) stored announcement $\sigma_{W_j}(M, close)$ and stops signing new state updates.

`/* Closing party submits also proofs-of-fraud */`

3. Party p, upon verifying t on-chain signed announcements by the wardens, recovers the $max(i)$ that is included in the announcements. Then, party p publishes on-chain the state s_i, the random number r_i, the signature of both parties on $\{H(s_i, r_i), i\}$, and any proofs-of-fraud.

`/* Closing the channel with punishments */`

4. After the state is included in a (permanent) block, the smart contract recovers $\{H(s_i, r_i), i\}$, and verifies both parties' signatures, the wardens identities, and the proofs-of-fraud.

(a) If the valid proofs-of-fraud $x \leq f$, the smart contract closes the channel as soon as $t + x$ wardens have published an announcement on-chain. The channel closes in the state with the maximum sequence number included in the announcements, s_i. `// Protocol 5 with` $t + x$ `wardens`

(b) If the valid proofs-of-fraud $x \geq f + 1$, the smart contract closes the channel, and awards the entire channel balance to the counterparty.

The smart contract awards the collateral of cheating wardens to party p, and returns the collateral of all non-cheating wardens. The first t non-cheating wardens whose signature are published on-chain get an equal fraction of the closing fee F/t.

We further demand that the size of the committee is at least $n > 7$, hence $f > 2$. As a result, we guarantee there is at least one channel party with locked funds greater than each individual warden's collateral, $\frac{v}{2} > \frac{v}{f}$. This restriction along

with the aforementioned incentive mechanisms ensure resistance to collusion and bribing of the committee, meaning that following the protocol is the dominant strategy for the rational wardens.

We note that in a network with multiple channels, each channel needs to maintain a unique id which will be included in the announcement to avoid replay attacks. Otherwise, if there exist two channels with the same parties and watchtowers, the parties can unjustly claim the watchtowers' collateral by using signed sequence numbers from the other channel, effectively violating safety.

4 Brick Analysis

We first prove BRICK satisfies *safety* and *liveness* assuming at least one honest channel party and at least t honest wardens. Furthermore, we note that BRICK achieves *privacy* even if all wardens are byzantine while the channel parties are rational. Then, we show that rational players (parties and wardens) that want to maximize their profit will follow the protocol specification, for the incentive mechanisms presented in Section 3.2. Essentially, we show that BRICK enriched with the proposed incentive mechanisms is dominant-strategy incentive-compatible.

4.1 Security under one Honest Participant and t Honest wardens

We first show that closing a BRICK channel is safe in asynchrony when at least t wardens are honest while the rest are byzantine. The core idea is that the channel will close in the state that corresponds to the maximum sequence number submitted by t wardens. In particular, every transaction is broadcast to the wardens and confirmed by at least t wardens before the transaction is executed by the parties. Given that at most f wardens are byzantine and at most another $n-t = 3f+1-(2f+1) = f$ can be slow, then at most $2f$ can publish an outdated sequence number when closing the channel. Since BRICK waits for $t = 2f + 1$ sequence numbers, at least one will be submitted by an honest and up-to-date warden which will bear the maximum sequence number and correspond to the freshest state.

Theorem 1. BRICK *achieves safety in asynchrony assuming one byzantine party and f byzantine wardens.*

Proof. In BRICK there are two ways to close a channel (phase *Close*), either by invoking Protocol 4 or by invoking Protocol 5. In the first case (Optimistic Close), both parties agree on closing the channel in a specific state (which is always the freshest valid state[9]). As long as this valid state is published in a block in the persistent part of the blockchain, it is considered to be committed. Thus, safety is guaranteed.

In the second case, when Protocol 5 (Pessimistic Close) is invoked, a party has decided to close the channel unilaterally in collaboration with the committee.

[9] We assume that if the parties want to close the channel in a previous state, they will still create a new state similar to the previous one but with an updated sequence number.

The Brick smart contract verifies that the state is valid, i.e., the signatures of both parties are present and the sequence number of that state is the maximum from the submitted announcements. Given the validity of the closing state, it is enough to show that the channel cannot close in a state with a sequence number smaller than the one in the freshest committed state. This holds because even if the channel closes in a valid but not yet committed state with sequence number larger than the freshest committed state, this state will eventually become the freshest committed state (similarly to Protocol 4).

Let us denote by s_i the closing state of the channel. Suppose that there is a committed state s_k such that $k > i$, thus s_i is not the freshest state agreed by both parties. We will prove safety by contradiction. For the channel to close at s_i at least $t = 2f + 1$ wardens have provided a signed closing announcement, and the maximum sequence number of these announcements is i. Otherwise the Brick smart contract would not have accepted the closing state as valid. According to the threat model, at most $n - t = f$ wardens are byzantine, thus at least $f + 1$ honest wardens have submitted a closing announcement. Hence, none of the $f + 1$ honest wardens have received and signed the announcement of any state with sequence number greater than i. However, in phase *Update*, an update state is considered to be committed, according to Protocol 3 (step 3), if and only if it has been signed by at least $t = 2f + 1$ wardens. Since at most $n-(f+1) = 3f+1-f-1 = 2f < 2f+1$ wardens have seen (and hence signed) the state s_k, the state s_k is not committed. Contradiction.

We note that safety is guaranteed even if both parties crash. This holds because a state update requires unanimous agreement between the parties of the channel, i.e., both parties sign the hash of the new state. □

Note that a channel can close in two possible states: either the last agreed state by both parties, or the previous one. We still preserve safety in both cases. If the last agreed state is considered valid then it is guaranteed to be the closing state, whereas if the closing state is the previous then the last agreed state never gets validated by t wardens.

Theorem 2. Brick *achieves liveness in asynchrony assuming one byzantine party and f byzantine wardens.*

Proof. We will show that every possible valid operation is either committed or invalidated. There are two distinct operations: *close* and *update*. We say that operation close applies in a state s if this state was published on-chain either in collaboration of both parties (Protocol 4) or unilaterally by a channel party as the closing state (step 3, Protocol 5).

If the operation is close and not committed, either the parties did not agree on this operation (Optimistic Close), or a verification of the smart contract failed (Pessimistic Close). In both cases, the operation is not valid.

Suppose now the operation is close and never invalidated. Then, if it is an optimistic close, all the parties of the channel have signed the closing state since it is valid. Since at least one party is honest the transaction will be broadcast to the blockchain. As soon as the blockchain is live, the state will be included

in a block in the persistent part of the blockchain, and thus, the state will be eventually committed. On the other hand, if it is a pessimistic close and not invalidated, the smart contact verifications were successful therefore the state was committed.

Suppose the operation is a valid update and it was never committed. Since the operation is valid and at least one party of the channel is honest, the wardens eventually received the state update (Consistent Broadcast). However, the new state was never committed, therefore at least $f + 1$ wardens did not sign the update state. We assumed at most f byzantine wardens, hence at least one honest warden did not sign the valid update state. According to Protocol 3 (line 2), an honest warden does specific verifications and if the verifications hold the warden signs the new state. Thus, for the honest warden that did not sign, one of the verifications failed. If the first verification fails, then a signature from the parties of the channel is missing thus the state is not valid. Contradiction. The second verification concerns the sequence number and cannot fail, assuming at least one honest channel party. Thus, the warden has published previously a closing announcement on-chain and ignores the state update. In this case, either (a) the closing state of the channel is the new state - submitted by another warden that received the update before the closing request - or (b) the closing state had a smaller sequence number from the new state. In the first case (a), the new state is committed eventually (on-chain), while in the second case (b) the new state is invalidated as the channel closed in a previous state.

For the last case, suppose the operation is a valid update and it was never invalidated. We will show the state update was eventually committed. Suppose the negation of the argument towards contradiction. We want to prove that an update state that is not committed is either not valid or invalidated. The reasoning of the proof is similar to the previous case. □

Lastly, BRICK achieves *privacy even against byzantine wardens*. They only receive the sequence number of each update. Therefore, as long as parties do not intentionally reveal information, privacy is maintained.

4.2 Incentivizing Rational Players

In this section, we show that rational players, parties and wardens, that want to maximize their profit follow the protocols, i.e., deviating from the honest protocol executions can only result in decreasing a player's expected payoff. Therefore, security and liveness hold from Theorems 1 and 2. Note that in our system model $2f + 1$ wardens and the richest party are rational, while the rest can be byzantine. We consider each protocol separately, and evaluate the players' payoff for each possible action.

Intuitively, we provide correct incentives without utilizing timelocks because the closing party is the one penalizing the cheating wardens when closing the channel. Although counter-intuitive, the main idea is that the cheating party that convinced the wardens to cheat will profit more from collecting the collateral of the cheating wardens than closing the channel in any old state. Or in other words, the cheating party is actually rationally baiting the possible byzantine wardens

to get their collateral. As we show, this leads to rational wardens following the protocol faithfully.

Open. Naturally, if a party is not incentivized to open a channel then that channel will never be opened. We assume the parties have some business interest to use the blockchain and since transacting on channels is faster and cheaper they will prefer it. Deviating from the protocol at this phase is meaningless. Furthermore, we assume the wardens follow Protocol 1 and commit the requested collateral on-chain (BRICK smart contract). Otherwise, the parties simply replace the unresponsive/misbehaving wardens.

Update. In phase *Update*, we analyze Protocol 2 for rational parties as the wardens do not participate in this protocol. Then, we analyze Protocol 3 for both parties and wardens.

Lemma 1. *Rational parties faithfully follow Protocol 2.*

Proof. During the execution of Protocol 2, any party can deviate from the protocol by not signing the hash of the new state. In this case, the new state will not be valid and thus cannot be committed and will not be executed. No party can increase its profit from such behavior directly (attacking the safety of the channel). The same argument holds in case the party signs the hash but not the sequence number. Moreover, attempting to attack the liveness of the channel is not profitable since the counterparty can always request to close in collaboration with the committee by invoking Protocol 6.

Lastly, channel parties can collude and stop updating the channel (liveness attack) in order to enforce a hostage situation on the wardens' collateral. However, the committee size is at least $n > 7$. Thereby, from the pigeonhole principle there is at least one party in the channel that has locked funds at least equal to $\frac{v}{2} > \frac{v}{f}$, i.e., the richest party of the channel. Thus, the richest channel party locks an amount larger than each warden's collateral which means that this party's cost is larger than a wardens. Since we assume the richest party is rational, at any time this party is incentivized to close the channel in collaboration with the committee. Thus, the richest party (if rational) will not deviate from the honest execution of Protocol 2. □

Lemma 2. *Rational parties faithfully follow Protocol 3.*

Proof. During the execution of Protocol 3, a party can deviate as follows:

(a) First, a party can choose not to broadcast the announcement to the committee or part of the committee. In this case, the party has signed the new state, which is now a valid state. This state will be considered committed for the counterparty after the execution of Protocol 3. We show a rational party cannot increase its payoff by not broadcasting the announcement to all wardens. To demonstrate this, we consider two cases; either the new state is beneficial to the party or not. If the new state is beneficial to party A, then this state is not beneficial for the counterparty (e.g., B payed A for a service). Thus, if the committee has not received the freshest state, party B

can "rightfully" close the channel in the previous state. Hence, the expected payoff of party A decreases. On the other hand, suppose the new state is not beneficial to party A and it chooses not to send the announcement to the committee. Then, either the counterparty will have the state committed or the state will not be executed. From the safety property of the channels, party A cannot successfully close the channel in a previous state if the state was committed. Hence, party A does not increase its payoff. On the other hand, if party A does not request the signature of a warden that will later commit fraud, then the party cannot construct a proof-of-fraud to claim the warden's collateral and therefore the party's payoff decreases.

(b) Second, a party can broadcast different messages to the committee or parts of the committee. During the execution of Protocol 3, the wardens verify the parties' signatures, thus an invalid message will not be acknowledged from an honest warden. If the messages are valid (both parties' signatures are present), the parties have misbehaved in collaboration. This can lead to a permanent partition of the view of the committee regarding the state history, but at most one of the states can be committed (get the $2f + 1$ signatures). Thus, this strategy has the same caveats as the previous one, where the party can only lose from following it.

(c) Lastly, the party can choose not to proceed to the state transition. This is outside the scope of this work and a problem of a different nature (a fair exchange problem). □

Lemma 3. *Rational wardens faithfully follow Protocol 3.*

Proof. A warden only acts in step 2 of Protocol 3, and can deviate as follows:

(a) The warden does not perform the necessary verifications (signatures and sequence number). Then, the warden might unintentionally commit fraud by signing an invalid state, hence allow the party to claim the collateral.

(b) The warden replies to the party although it has published a closing announcement on-chain. Then, the party can penalize the warden by claiming its collateral.

(c) The warden sends its signature on the new state but does not store the new announcement. In this case, if a party requests to close unilaterally the warden cannot participate and thence collect the closing fee. Consequently, the warden's expected payoff decreases.

(d) The warden ignores the party's request to update the state (does not reply to the party), since the update fee is already collected. At first sight, this game looks like a fair exchange game, which is impossible to solve without a trusted third party [17]. Furthermore, we cannot use a blockchain to solve it [34] as the whole point of channels is to reduce the number of transactions that go on-chain. Fortunately, the state update game is a repeated game where wardens want to increase their expected rewards in the long term. The wardens know that if they receive an update fee from a party and do not respond, then the party will stop using them (there is f fault tolerance in BRICK), thus their expected payoff for the repeated game will decrease. □

Close. A channel can either close optimistically by both parties with Protocol 4, or unilaterally by one of the parties in collaboration with the wardens with Protocol 6. We first show that rational parties do not deviate from Protocol 4 (Lemma 4), and later we consider the most complicated case of Protocol 6. Intuitively, a rational party invoking Protocol 6 could pretend to "cheat" and collect proofs-of-fraud from the wardens. *However, the rational party always prefers to close the channel in the correct state and claim the wardens' collateral, when $C \geq v/f$, as they sum up to larger profits (Lemma 5).* The rational wardens, therefore, will not collaborate with a cheating party but instead follow Protocol 5. As a result, Brick channels achieve safety and liveness with rational players (Theorem 3).

Lemma 4. *Rational parties faithfully follow Protocol 4.*

Proof. A party can deviate from Protocol 4 in the following ways:

(a) It is the party requesting the closing of the channel in a cheating state. The counterparty will not sign the state since it is being cheated, else it would not be a cheating state. Thus, safety is guaranteed and the party cannot profit from this strategy.
(b) It is the party requesting the closing of the channel and never publishes the signed closing state. In line 2 of Protocol 4, the signatures on the state are exchanged between the parties, hence the counterparty will eventually publish the closing state. Note that we assume that the closing party sends its signature first with the closing request.
(c) It is the party that got the closing request and does not sign the state. In this case, the party requesting to close the channel can invoke Protocol 6 and close the channel in collaboration with the committee in the freshest committed state. □

Lemma 5. *Rational parties invoking Protocol 6 maximize their profit when closing the channel in the freshest committed state.*

Proof. Let us denote by p_A the payoff function of party A (the cheating party wlog). The payoff function depends on the channel balance of party A when requesting to close the channel, denoted by c_A ($0 \geq c_A \geq v$, where v is the total channel funds), the collateral the party claims through proofs-of-fraud, and the total amount spent for bribing rational wardens. Formally,

$$p_A = c_A + x\frac{v}{f} - b\big(\frac{v}{f} + \epsilon\big)$$

where x is the number of proofs-of-fraud submitted by the party, b the number of bribed rational wardens, and ϵ the marginal gain over the collateral the wardens require to be bribed. Note that each warden has locked $\frac{v}{f}$ as collateral. Further, note that the byzantine wardens do not require a bribe but act arbitrarily malicious, meaning that they will provide a proof-of-fraud to party A without any compensation.

Next, we analyze all potential strategies for party A and demonstrate that the payoff function maximizes when the party does not bribe any rational warden, but closes the channel in the freshest committed state. There are four different outcomes in the strategy space of party A:

(a) The channel closes in the freshest committed state and no proofs-of-fraud are submitted. Then, $p_A = c_A$.
(b) Party A submits the proofs-of-fraud only from the byzantine wardens and the channel closes in the freshest committed state (by the remaining t rational warden). Then, $p_A = c_A + f\frac{v}{f} = c_A + v$. Note that the payoff function in this case maximizes for $x = f$.
(c) Party A bribes $b > 0$ rational wardens and the channel closes in the freshest committed state. Then, $p_A \leq c_A + (f+b)\frac{v}{f} - b(\frac{v}{f} + \epsilon) = c_A + v - b\epsilon \leq c_A + v - \epsilon$. The first inequality holds because the bribed wardens might be part of the second set of t closing announcements, in which case they do not contribute to the claimed collateral.
(d) Party A bribes $b > 0$ rational wardens and the channel closes in a state other than the freshest committed state[10]. Let us denote by y the number of rational wardens that provide a proof-of-fraud to the party, and m the number of submitted proofs-of-fraud that belong to byzantine wardens. Then, the possible actions in this strategy are depicted in Table 1. Note that at least $f + 1$ misbehaving wardens are required to close the state in a previous state since we assume there can be f slow rational wardens that have not yet received the update states. In this case, the payoff function is

Table 1: Potential actions to close the channel in a "fraudulent" state. The first column illustrates the wardens for which the closing party submits proofs-of-fraud; y denotes the rational wardens and m the byzantine. The second column depicts the wardens for which no proof-of-fraud was submitted, hence they "count" for closing the channel. Since f rational wardens may be slow in asynchrony, the closing party needs $f + 1$ submitted sequence numbers from wardens that "count" to close the channel.

Action	**Proof-of-fraud**	**Close**	**Total**
Byzantine	m	$f - m$	f
Bribed (rational)	y	$f + 1 - (f - m)$ $= m + 1$	$y + m + 1$
Total	$m + y$	$f + 1$	-

$$p_A \leq v + \big(m + y\big)\frac{v}{f} - \big(y + m + 1\big)\big(\frac{v}{f} + \epsilon\big)$$

$$= v - \frac{v}{f} - \epsilon\big(y + m + 1\big) \leq v - \frac{v}{f} - \epsilon$$

[10] Note that b varies between 1 and $f + 1$. That is because up to f wardens may be slow in asynchrony, hence might provide truthful confirmation of an old state to the closing party.

where the first inequality holds since $0 \geq c_A \geq v$. Therefore, the payoff function maximizes in case the party follows the second strategy, i.e., when the channel closes in the freshest committed state and no rational wardens are bribed. □

Using Lemma 5, we show that rational parties follow Protocol 5 with respect to the rational wardens. That is, apart from the proofs-of-frauds submitted by the party for the byzantine wardens that "forfeit" their collateral to the closing party, in every other way, the selfish behavior of the closing party executing Protocol 6 aligns with the honest behavior of a party executing Protocol 5.

Lemma 6. *Rational parties are incentivized to faithfully follow Protocol 5 with respect to the rational wardens.*

Proof. The party that requested to close, invoking Protocol 5, can deviate from the protocol's specification in two ways: either (a) the party publishes an invalid closing state (e.g., random state, previously valid state, a committed state that is not the freshest), or (b) the party is not responsive, meaning that the party does not publish the closing state.

In case (a), publishing an invalid state can only decrease a party's profit, as shown in Lemma 5.

In case (b), either the party is the richest of the channel or not. If the party is the richest of the two then the party's cost of not responding is higher than that of the counterparty and any other warden's cost. Therefore, the party is not the richest of the channel. In this case, the counterparty which wants to close the channel, can use the on-chain closing announcements of the wardens and publish the closing state. In both cases, the party that requested close cannot increase its payoff by not revealing the closing state, but only lose from locking its channel funds for a longer period of time that necessary (since no updates are possible). □

Next, we show that if the smart contract executes Protocol 6 when closing the channel, the rational wardens will honestly follow Protocol 5; implying that the wardens will not commit fraud in collaboration with the closing party.

Lemma 7. *Rational wardens faithfully follow Protocol 5.*

Proof. A warden can deviate from the protocol as follows:

(a) The warden does not publish a closing announcement on-chain. In this case, the warden can attempt to enforce a hostage situation on the funds of the channel in collaboration with other wardens in order to blackmail the channel parties. However, to enforce a hostage situation on the channel's funds, at least $f + 1$ wardens must collude, hence at least one rational warden must participate. However, a rational warden cannot be certain that the other wardens will indeed maintain the hostage situation or participate in the consensus thus claim the closing fee (only the first t wardens get paid); therefore, we reduce our problem to the prisoner's dilemma problem. As a

result, the only strong Nash equilibrium for a rational warden is to immediately publish a closing announcement on-chain in order to claim later the closing fee.

(b) The warden signs later a new state update. Then, the warden allows the party receiving the signed announcement with higher sequence number than the closing announcement to create a proof-of-fraud and claim the warden's collateral.

(c) The warden publishes on-chain a closing announcement that is not the stored one[11]. However, the warden has already sent a signature on an announcement with a higher sequence number, therefore the party can create a proof-of-fraud and claim the warden's collateral. Hence, any rational warden will request as a bribe an amount at least marginally higher than the collateral to perform the fraud. In Lemma 5, we show that no rational party will provide such a bribe to any rational warden. Thus, all rational wardens will honestly follow the protocol and submit the stored announcement for closing the channel. □

Theorem 3. BRICK *channels achieve safety and liveness in asynchrony assuming the richest party and* $2f+1$ *wardens are rational, while the rest are byzantine (wardens and other party).*

Proof. We showed that any rational player, party or warden, honestly follows Protocols 1, 2 (Lemma 1), 3 (Lemmas 2 and 3), 4 (Lemma 4), and 5 (Lemmas 6 and 7). Therefore, both safety and liveness are achieved in asynchrony in our system model from Theorems 1 and 2, respectively. □

5 Evaluation of Brick

In this section we evaluate both the cost of consistent broadcast as well as the cost of BRICK's on-chain operations. The key questions we want to answer are: (a) How does the cost of deploying BRICK change as we increase the number of wardens (i.e., the security and fault tolerance), and (b) how does the cost of the off-chain part scale now that we need to interact with the wardens every time in order to protect against network attacks.

Solidity Smart Contract. To evaluate the cost of deployment of BRICK on Ethereum, we have implemented the on-chain operations in the form of a smart contract in Solidity[12]. We measured the gas cost of deploying and operating the smart contract on the Ethereum blockchain. Our results are illustrated in Figure 1. Our gas measurements are conducted using prices as of May 2020, namely the fiat price of 1 ETH = 195.37 EUR and a gas price of 20 Gwei (for convenience, prices on the diagram are shown in both EUR and Ether). The

[11] We assume the warden will only sign as closing an announcement that used to be valid in an attempt to commit fraud. Otherwise, the party will claim the warden's when the smart contract verifications fail.

[12] The source code of the smart contract is released under the open source MIT license and is available anonymously at `https://github.com/dionyziz/brick`.

contract was implemented in Solidity 0.5.16 and measurements were performed using using the solc 0.6.8 compiler with optimizations enabled, deployed on a local ganache-cli blockchain using truffle and web3. The measurements concern the deployment of the smart contract, the opening of the channel, optimistically closing the channel, and pessimistically closing the channel. The contract allows the parties to specify the number n of wardens they desire to involve as well as their identities. To aid with the EVM implementation, we opted for the `secp256k1` elliptic curve signature scheme [11, 23], as the signatures generated by it can be verified in a gas-efficient manner in Solidity using the `ecrecover` precompiled smart contract [40]. Additionally, the signature scheme makes off-chain signatures compatible with on-chain accounts and as such signatures made off-chain can be verified using public keys available on-chain. We measured the gas cost for warden values of $n = 3$ to 30. We recommend the value of $n = 13$ as highlighted in the figure, since it is safe, has good performance and aligns the incentives correctly.

Once the contract is deployed on the Ethereum network, Alice funds it first. Subsequently, once Alice's funding transaction is finalized, Bob funds it. Once Bob's funding transaction is finalized, the collateral can be calculated and so the wardens can fund it in any order simultaneously. When all wardens have funded the contract, any of the two parties can open the channel. At any time prior to opening the channel, any party or warden can withdraw their money, at which point the channel is cancelled and can no longer be opened, but allows the rest of the parties to withdraw as well, in any order. Once the channel is open, the parties can continue exchanging states off-chain. If multiple Brick channels are used, then the cost of smart contract deployment can be amortized over all of them by abstracting the common functionality into a Solidity library. However, the opening and closing costs are recurrent. We remark here that our cost of deployment ($\approx$ 9 EUR) is comparable to other state channel smart contracts which perform different operations under different assumptions (e.g., at current prices, the deployment of the 3 Pisa [30] contracts amounts to $\approx$ 17 EUR).

When the parties wish to close the channel optimistically, initially Alice submits a transaction to the smart contract requesting the channel to close. This request contains Alice's claimed closing state (namely, Alice's value at closing time, as Bob's value at closing time can be deduced from this). Once Alice's transaction is confirmed, if Bob is in agreement, he submits a transaction to the smart contract to signal his agreement. The smart contract then returns Alice's and Bob's values as well as wardens' collateral. Care must be taken to check that the sum of Alice's value and Bob's value at the closing state does not exceed the sum of their values at their initial state, so that sufficient funds remain to return the wardens' collateral. If Bob does not agree with Alice's claim, the channel becomes unusable and must be closed pessimistically (Alice can no longer make an optimistic claim on a different state). The optimistic close operation measures the cumulative gas cost of the two transactions from both parties. The cost is minimal and should be the normal path since parties need not pay closing fees.

Finally, the channel can be closed pessimistically at any time. To do this, the party who wishes to close the channel requests this from the wardens. Each warden then submits a transaction to the smart contract containing the sequence number they have last seen, together with Alice's and Bob's off-chain signatures on it. These can be submitted in any order. The signatures of Alice and Bob on the sequence number are verified on-chain, along with the warden signatures; this incurs the majority of gas cost for the pessimistic close. The party who wishes to close the channel monitors the chain for such claims and remembers any of them that are fraudulent. As soon as t (honest or adversarial) claims have been recorded, either Alice or Bob can send a transaction to the smart contract to close the channel. The transaction is accompanied by the fraud proofs the closing party was able to assemble, namely the latest announcement for each warden who made a claim on an earlier sequence number. These announcements contain the signature of the warden on the plaintext which consists of the smart contract address and the sequence number. As smart contracts have unique addresses, including the smart contract address in the plaintext ensures that the same warden can participate in multiple BRICK channels simultaneously using the same public key[13]. Closing the channel releases the funds of the participant parties and slashes any malicious wardens. After the channel has been closed, any honest wardens who wish to redeem their collateral and their corresponding fee can do so by issuing a further transaction to the smart contract. The pessimistic close operation was measured when no fraud proofs are provided and includes the transaction of each of the t wardens and the final transaction by one of the parties. Additionally, it was assumed that, while the counterparty is unresponsive or malicious, the wardens were responsive and all submitted the same sequence number (hence limiting the need for multiple signature validations).

Consistent Broadcast. We have also implemented consistent broadcast in Golang using the Kyber [27] cryptographic library and the cothority [14] framework. In Table 2 we evaluate our protocol on Deterlab [15] using 36 physical machines, each having four Intel E5-2420 v2 CPUs and 24 GB RAM. To have a realistic wide area network, we impose a 100ms roundtrip latency on the links between wardens and a 35Mbps bandwidth limit.

As illustrated the overhead of using a committee is almost equal to a round-trip latency (100ms). The small overhead is due to the party sending the messages in sequence, hence the last message is sent with a small delay $d > 0$. This is observed in the total latency which is close to $100 + d$ ms. This latency in BRICK defines the time parties must wait to execute a transaction safely, meaning that BRICK provides fast finality. These numbers are three orders of magnitude faster than current blockchains, such as Ethereum where blocks are generated on average every 12sec and finality is guaranteed after 6 minutes. Furthermore, channels are independent and embarrassingly parallel which means that we can deploy as many as we want without significantly increasing the overhead. In contrast

[13] If BRICK is deployed on multiple alt-etherea, each warden must use a different key in each, or the smart contract must be modified to have the warden sign the `CHAIN_ID` together with the address and sequence number.

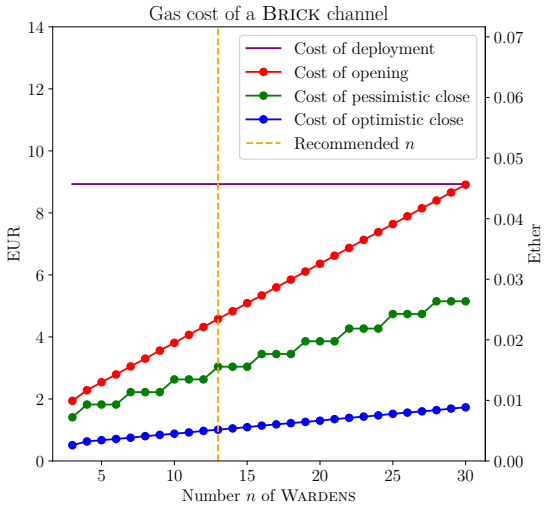

Fig. 1: The on-chain gas cost of deploying and operating Brick in the form of an Ethereum smart contract.

to synchronous channel solutions where finality under our model is guaranteed only after channel closure, Brick provides fast finality (rtt).

Table 2: Microbenchmark of Brick

Number of wardens	**7**	**34**	**151**
Consistent Broadcast	0.1138 sec	0.118 sec	0.1338 sec

6 Related Work

Payment channels were originally introduced by Spilman [38]. Several payment channels solutions have been proposed [1,13,35], the most notable being the Bitcoin Lightning Network [35]. All these solutions, however, require timelocks to guarantee safety and therefore make strong synchrony assumptions that sometimes fail in practice.

To guarantee safety, traditional payment channels require participants to be frequently online, actively watching the blockchain. To alleviate this necessity, recent proposals introduced third-parties in the channel to act as proxies for the participants of the channel in case of fraud. This idea was initially discussed by Dryja [16], who introduced the Monitors or Watchtowers [2] operating on the Bitcoin Lightning network [35]. Later, Avarikioti et al. proposed DCWC [5], a less centralized distributed protocol for the watchtower service, where every full node can act as a watchtower for multiple channels depending on the network topology. In both these works, the watchtowers are paid upon fraud. Hence, the solutions are not incentive-compatible since in case a watchtower is employed no rational party will commit fraud on the channel and thus the watchtowers will

never be paid. This means there will be no third parties offering such a service unless we assume they are altruistic.

In parallel, McCorry et al. [30] proposed Pisa, a protocol that enables the delegation of Sprites [32] channels' safety to third-parties called Custodians. Although Pisa proposes both rewards and penalties similarly to Brick, it fails to secure the channels against bribing attacks. Particularly, the watchtower's collateral can be double-spent since it is not tied to the channel/party that employed the watchtower. More importantly, similarly to Watchtowers and DCWC, Pisa demands a synchronous network and a perfect blockchain, meaning that transactions must not be censored, to guarantee the safety of channels.

Concurrently to this work, Avarikioti et al. introduced Cerberus channels [7], a modification of Lightning that incorporates rational watchtowers to Bitcoin channels. Although Cerberus channels are incentive-compatible, they still require timelocks, hence their security depends on synchrony assumptions and a perfect blockchain that cannot be censored. Furthermore, Cerberus channels do not guarantee privacy from the watchtowers, as opposed to Brick.

In a similar work, Lind et al. proposed Teechain [28], a layer 2 payment network that operates securely under asynchrony using hardware trusted execution environments (TEEs) to prevent parties from misbehaving. In contrast, Brick eliminates the need for TEEs with the appropriate incentive mechanisms.

To summarize, we exhibit the differences of Brick to the other channel constructions and watchtower solutions in Table 3. We observe that Brick is the only solution that maintains security under an asynchronous network and offline channel parties while assuming rational watchtowers. Further, Brick is secure (i.e., no loss of funds) even when the blockchain substrate is censored, and also when the network is congested. Finally, an extension to Brick that we describe in Section 7 enables protection against small scale persistence attacks making it more secure than the underlying blockchain.

Table 3: Comparison with previous work

Protocol	Monitors	DCWC	Pisa	Cerberus	Brick
Safe under	[16]	[5]	[30]	[7]	[6]
Rational Players	✗	✗	∼[14]	✓	✓
Offline Parties	✓	✓	$T \gg t_d$[15]	$T \gg t_d$[15]	✓
Asynchrony	✗	✗	✗	✗	✓
Censorship	✗	✗	✗	✗	✓
Congestion	✗	✗	✗	✗	✓
Forks	✗	✗	✗	✗	✓[16]
Privacy	✓	✓	✓	✗	✓
Bitcoin Compat.	✓	✓	✗	✓	✗

State Channels, Payment Networks, and Sidechains. Payment channels can only support payments between users. To extend this solution to handle smart contracts [39] that allow arbitrary operations and computations, *state channels* were introduced [32]. Recently, multiple state channel constructions have emerged [10, 18, 32]. However, all these constructions use the same foundations, i.e., the same concept on the operation of two-party channels. And as the fundamental channel solutions are flawed the whole construction inherits the same problems (synchrony and availability assumptions). Brick's design could potentially extend to an asynchronous state channel solution if there existed a valuation function for the states of the contract (i.e., a mapping of each state to a monetary value for the parties) to correctly align incentives. In this case, the channel can evolve as long as the parties update the state, while in case of of an uncooperative counterparty the honest party can always pessimistically close the channel at the last agreed state and continue execution on-chain.

Another solution for scaling blockchains is sidechains [8, 20, 25]. In this solution, the workload of the main chain is transferred in other chains, the sidechains, which are "pegged" to the main chain. Although the solution is kindred to channels, it differs significantly in one aspect: in channels, the states updates are totally ordered and unanimously agreed by the parties thus a consensus process is not necessary. On the contrary, sidechains must operate a consensus process to agree on the validity of a state update. Brick lies in the intersection of the two concepts; the states are totally ordered and agreed by the parties, whereas wardens merely remember that agreement was reached at the last state announced.

Finally, an extension to payment channels is payment channel networks (PCN) [9, 29, 36, 37]. The core idea of PCN is that users that do not have a direct channel can route payments using the channels of other users. While Brick presents a novel channel construction that is safe under asynchrony, enabling asynchronous multi-hop payments remains an open question.

7 Conclusion, Limitations and Extensions

Below, we discuss the rationale of Brick design, its limitations, and possible extensions.

Byzantine players. If both channel parties are byzantine then the wardens' collateral can be locked arbitrarily long since the parties can simply crash forever. This is why in the threat model, we assume that at least the richest channel party is rational to correctly align the incentives. We further demand byzantine fault-tolerance to guarantee a truly robust protocol against arbitrary faults. We assume at most f out of the $3f + 1$ wardens are byzantine, which is necessary to maintain safety, as dictated by well known lower bounds for asynchronous

[14] The watchtower needs to lock collateral per-channel, equal to the channel's value. Current implementation of Pisa does not provide this.

[15] The party needs to be able to deliver messages and punish the watchtower within a large synchrony bound T.

[16] Possible if consensus is run for closing the channel as described in Section 7.

consistent broadcast. Nevertheless, users of BRICK can always assume $f = 0$ and configure the smart contract parameters accordingly.

Warden unilateral exit. If both parties are malicious, they might hold the wardens' collateral hostage. A similar situation is indistinguishable from the parties not transacting often. As a result the wardens might want to exit the channel. A potential extension can support this in two ways. First, we can enable committee replacement, meaning that a warden can withdraw its service as long as there is another warden willing to take its place. In such a case, we simply replace the collateral and warden identities with an update of the funding transaction on-chain, paid by the warden that requests to withdraw its service. Second, if a significant number (e.g, $2f + 1$) of wardens declare they want to exit, the smart-contract can release them and convert the channel to a synchronous channel [30]. The parties will now be able to close the channel unilaterally by directly publishing the last valid state. If the counterparty tries to cheat and publishes an old state, the party (or any remaining warden) can catch the dispute on-time and additionally claim the (substantial) closing fee.

Committee selection. Each channel has its own group of wardens, i.e., the committee is independently selected for each channel by the channel parties. The scalability of the system is not affected by the use of a committee since each channel has its own independent committee of wardens. The size of the committee for each channel can vary but is constrained by the threat model. If we assume at least one honest party in the channel, a single rational warden is enough to guarantee the correct operation of BRICK. Otherwise, we require more than 7 wardens to avoid hostage situations from colluding channel parties (Section 3.2). Note that the cost for security for the parties is not dependent on the committee size, but on the value of the channel. If the parties chose a small committee size, the collateral per warden is high, thus the update fees are few but high. On the other hand, if the parties employ many wardens, the collateral per warden is low, thus the update fees are many but low.

Consensus vs consistent broadcast. Employing consistent broadcast in a blockchain system typically implies no conflict resolution as there is no liveness guarantee if the sender equivocates. This is not an issue in channels since a valid update needs to be signed by both parties and we provide safety guarantees only to honest and rational parties[17]. The state updates in channels are totally ordered by the parties and each sequence number should have a unique corresponding state. Thereby, it in not the role of the warden committee to enforce agreement, but merely to verify that agreement was reached, and act as a shared memory for the parties. As a result, consistent broadcast is tailored for BRICK as it offers the only necessary property, equivocation protection.

Brick Security under execution fork attacks. We can extend BRICK to run asynchronous consensus [26] during the closing phase in order to defend against execution fork attacks [24]. This would add an one-off overhead during close but would make BRICK resilient against extreme conditions [4]. For example, in case

[17] Of course if a party crashes we cannot provide liveness, but safety holds.

of temporary dishonest majority the adversary can attack the persistence[18] of the underlying blockchain, meaning that the adversary can double-spend funds. Similarly in channels, if the adversary can violate persistence, the dispute resolution can be reversed, hence funds can be cheated out of a party. However, in Brick the adversary can only close on the last committed state or the freshest valid (not committed) state. With consensus during close, Brick maintains safety (i.e., no party loses channel funds) even when persistence is violated. A malicious party can only close the channel in the state that the consensus decides to be last, thus a temporary take-over can only affect the channel's liveness. Therefore, Brick can protect both against *liveness and persistence attacks*[19] on the underlying blockchain adding an extra layer of protection, and making it safer to transact on Brick than on the blockchain.

Update fees. Similarly to investing in stocks for a long period of time, many invest in cryptocurrencies; resulting in large amounts of unused capital. Acting as a warden can simply provide more profit (update fees) to the entities that own this capital complementary to owning such cryptocurrencies.

Currently, the update fees are awarded to wardens on every state update via a unidirectional channel. Ideally, these rewards would be included in the state update of the channel. But even if we include an increased fee on every state update, the parties can always invoke Optimistic Close, and update the channel state to their favor when closing. Thus, the incentives mechanism is not robust if the update rewards of the wardens are included in the state updates.

Collateral. The collateral for each warden in Brick is v/f, where v is the total value of the channel and f the number of byzantine wardens. This is slightly higher than the lowest amount $v/(f+1)$ for which security against bribing attacks is guaranteed in asynchrony when both channel parties and wardens are rational. Towards contradiction, we consider a channel where each warden locks collateral $C < v/(f+1)$. Suppose now a rational party p owns 0 coins in the freshest state and v coins in a previous state. Due to asynchrony, p controls the message delivery, hence f wardens may consider this previous state as the freshest one. Consequently, if p bribes $f+1$ wardens, which costs less than $(f+1)v/(f+1) = v$, the party profits from closing the channel in the previous state in collaboration with the bribed and "slow" wardens, violating safety.

In a synchronous network, this attack would not work since the other parties would have enough time to dispute. However, under asynchrony (or offline parties [30]) there is no such guarantee. Further, note that in a naive asynchronous protocol with f byzantine wardens, the previous attack is always possible for any collateral because a rational party can direct the profit from the collateral of the byzantine wardens to bribe the rational wardens. We circumvent this problem in Brick by changing the closing conditions; in particular, we force the closing

[18] Persistence states that once a transaction is included in the permanent part of one honest party's chain, then it will be included in every honest party's blockchain.

[19] We assume the channel to be created long before these attacks take place, so the adversary cannot fork the transaction that creates the channel.

party to choose between closing the channel in an old state, or claiming the collateral of at least $f + 1$ wardens and awarding the channel balance to the counterparty.

Finally, a trade-off for replacing trust is highlighted: online participation with synchrony requirements or appropriate incentive mechanisms to compel the honest behavior of rational players.

Decentralization. In previous payment channel solutions a party only hires a watchtower if it can count on it in case of an attack. Essentially, watchtowers are the equivalent of insurance companies. If the attack succeeds, the watchtower should reimburse the cheated channel party [30]. After all, it is the watchtower's fault for not checking the blockchain when needed. However, in light of network attacks (which are prevalent in blockchains [3, 21]), only a few, centrally connected miners will be willing to take this risk. Brick provides an alternative, that proactively protects from such attacks and we expect to provide better decentralization properties with minimal overhead and fast finality.

Bitcoin compatibility. We believe Brick can be implemented in Bitcoin assuming t honest wardens (Protocol 5) using chained transactions. In contrast, we conjecture that the incentive-compatible version of Brick (Protocol 6) cannot be deployed without timelocks in platforms with limited contracts like Bitcoin.

8 Acknowledgements

We would like to thank Kaoutar Elkhiyaoui for her valuable feedback as well as Jakub Sliwinski for his impactful contribution to this work.

References

1. Raiden network. `https://raiden.network/` (2017), accessed: 2020-11-22
2. Bitcoin Lightning Fraud? Laolu Is Building a 'Watchtower' to Fight It. `https://www.coindesk.com/laolu-building-watchtower-fight-bitcoin-lightning-fraud` (2018)
3. Apostolaki, M., Zohar, A., Vanbever, L.: Hijacking bitcoin: Routing attacks on cryptocurrencies. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 375–392. IEEE (2017)
4. Avarikioti, G., Käppeli, L., Wang, Y., Wattenhofer, R.: Bitcoin security under temporary dishonest majority. In: International Conference on Financial Cryptography and Data Security. pp. 466–483. Springer (2019)
5. Avarikioti, G., Laufenberg, F., Sliwinski, J., Wang, Y., Wattenhofer, R.: Towards secure and efficient payment channels. arXiv preprint: 1811.12740 (2018)
6. Avarikioti, Z., Kogias, E.K., Wattenhofer, R., Zindros, D.: Brick: Asynchronous incentive-compatible payment channels. In: International Conference on Financial Cryptography and Data Security (2021)
7. Avarikioti, Z., Litos, O.S.T., Wattenhofer, R.: Cerberus channels: Incentivizing watchtowers for bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 346–366. Springer (2020)
8. Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains. `https://www.blockstream.com/sidechains.pdf` (2014)

9. Bagaria, V., Neu, J., Tse, D.: Boomerang: Redundancy improves latency and throughput in payment networks. In: International Conference on Financial Cryptography and Data Security (2020)
10. Coleman, J., Horne, L., Xuanji, L.: Counterfactual: Generalized state channels. https://l4.ventures/papers/statechannels.pdf (2018)
11. Courtois, N.T., Grajek, M., Naik, R.: Optimizing sha256 in bitcoin mining. In: International Conference on Cryptography and Security Systems. pp. 131–144. Springer (2014)
12. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., Song, D., Wattenhofer, R.: On scaling decentralized blockchains. In: International Conference on Financial Cryptography and Data Security. pp. 106–125. Springer (2016)
13. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Stabilization, Safety, and Security of Distributed Systems. pp. 3–18. Springer (2015)
14. DeDiS cothority (2016), https://www.github.com/dedis/cothority
15. DeterLab network security testbed. http://isi.deterlab.net/ (2012)
16. Dryja, T.: Unlinkable outsourced channel monitoring. https://youtu.be/Gzg_u9gHc5Q (2016)
17. Dziembowski, S., Eckey, L., Faust, S.: Fairswap: How to fairly exchange digital goods. In: Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security. pp. 967–984. ACM (2018)
18. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: Virtual payment hubs over cryptocurrencies. In: IEEE Symposium on Security and Privacy. pp. 327–344 (2017)
19. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 281–310. Springer (2015)
20. Gaži, P., Kiayias, A., Zindros, D.: Proof-of-Stake Sidechains. In: IEEE Symposium on Security and Privacy. pp. 139–156. IEEE (2019)
21. Gervais, A., Ritzdorf, H., Karame, G.O., Capkun, S.: Tampering with the delivery of blocks and transactions in Bitcoin. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 692–705. ACM (2015)
22. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: International Conference on Financial Cryptography and Data Security. pp. 201–226. Springer (2020)
23. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.C.: Comparing elliptic curve cryptography and rsa on 8-bit cpus. In: International workshop on cryptographic hardware and embedded systems. pp. 119–132. Springer (2004)
24. Karame, G.O., Androulaki, E., Capkun, S.: Double-spending fast payments in Bitcoin. In: 19th ACM Conference on Computer and Communications Security. pp. 906–917. ACM (2012)
25. Kiayias, A., Zindros, D.: Proof-of-Work Sidechains. In: International Conference on Financial Cryptography and Data Security. pp. 21–34. Springer (2019)
26. Kokoris-Kogias, E., Malkhi, D., Spiegelman, A.: Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In: 27th ACM SIGSAC Conference on Computer and Communications Security. pp. 1751–1767. ACM (2020)
27. The Kyber Cryptography Library (2010 – 2018)

28. Lind, J., Naor, O., Eyal, I., Kelbert, F., Sirer, E.G., Pietzuch, P.R.: Teechain: a secure payment network with asynchronous blockchain access. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. pp. 63–79 (2019)
29. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M.: Silentwhispers: Enforcing security and privacy in decentralized credit networks. In: 24th Annual Network and Distributed System Security Symposium (2017)
30. McCorry, P., Bakshi, S., Bentov, I., Meiklejohn, S., Miller, A.: Pisa: Arbitration outsourcing for state channels. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 16–30. ACM (2019)
31. Miller, A.: Feather-forks: enforcing a blacklist with sub-50% hash power. `https://bitcointalk.org/index.php?topic=312668.0`, accessed: 2020-11-22
32. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: Payment networks that go faster than lightning. In: International Conference on Financial Cryptography and Data Security. pp. 508–526 (2019)
33. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
34. Pagnia, H., Gärtner, F.C.: On the impossibility of fair exchange without a trusted third party. Tech. rep., Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Germany (1999)
35. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2015)
36. Prihodko, P., Zhigulin, S., Sahno, M., Ostrovskiy, A., Osuntokun, O.: Flare: An approach to routing in lightning network (2016)
37. Roos, S., Moreno-Sanchez, P., Kate, A., Goldberg, I.: Settling payments fast and private: Efficient decentralized routing for path-based transactions. In: 25th Annual Network and Distributed Systems Security Symposium (2018)
38. Spilman, J.: Anti dos for tx replacement. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html`, accessed: 2020-11-22
39. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997)
40. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper (2014)
41. Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sanchez, P., Kiayias, A., Knottenbelt, W.J.: Sok: Communication across distributed ledgers. IACR Cryptology ePrint Archive, Report 2019/1128 (2019)