OptiBridge: A Trustless, Cost-Efficient Bridge Between the Lightning Network and Ethereum

Mohsen Minaei[0009-0001-3899-697X][1], Duc V. Le[0000-0002-8123-2713][1], and Pedro Moreno-Sanchez[0000-0003-2315-7839][1,2,3]

[1] Visa Research
[2] IMDEA Software Institute
[3] MPI-SP
{m.mohsen.minaei, levduc112}@gmail.com
pedro.moreno@imdea.org

Abstract. Bridges, protocols that enforce a state transition on a destination ledger conditioned on an event on a source ledger, are central to modern DeFi. Conventional designs implicitly assume the source ledger event is publicly observable, an assumption that breaks with Layer-2 payment channels such as the Lightning Network, where state updates occur off-chain between counterparties and are invisible to others. This state of affairs advocates for new bridge designs for this setting.
This paper introduces OptiBridge, a bridge between a payment channel (e.g., Lightning Network) and a smart-contract blockchain (e.g., Ethereum) that preserves safety and liveness without adding trust assumptions and remains fully compatible with existing Lightning and Ethereum stacks. OptiBridge follows an optimistic path in the common case: two honest channel peers materialize the intended state on the destination chain by revealing a pre-agreed secret. To handle faults and adversarial behavior, OptiBridge provides a dispute path orchestrated by a more expressive contract that is deployed *only on demand*. An implementation demonstrates substantial cost savings in the optimistic case: compared to Alba (NDSS'25), the optimistic contract deployment uses $\sim 73\%$ less gas (1.22M vs. 4.51M), and proof submission costs 40,107 vs. 253,566 gas; when disputes arise, the dispute contract deployment costs 2,785,514 gas and the core dispute call is cheaper (196,438 vs. 515,860). Our analysis shows that rational users strictly prefer the optimistic path, whereas the dispute mechanism prevents coin theft and imposes higher fees and delays on the deviator.

Keywords: Bridge · Optimistic · Ethereum · Lightning Network

1 Introduction

Numerous blockchains have been developed to address diverse use cases, and currently, over 100 blockchains are operational [2]. However, these blockchains generally lack native mechanisms for inter-chain communication. This limitation

is especially critical for decentralized finance (DeFi) applications—the flagship use case of modern blockchains—since it restricts the seamless transfer of assets across chains, resulting in fragmented liquidity and diminished market efficiency.

Bridges address this interoperability gap. Despite differences in their designs, most of them follow a standard lock-and-mint paradigm. In this paradigm, a user first locks assets on the source chain (step ①). Once the lock is confirmed, the bridge mints a corresponding amount of wrapped tokens on the destination chain (step ②) representing a 1:1 claim on the locked assets. To redeem, the user burns the wrapped tokens on the destination chain (step ③), after which the bridge releases the originally locked assets on the source chain (step ④).

The security properties of bridges are typically articulated in terms of *safety* and *liveness*. The safety property requires, first, that wrapped tokens on the destination chain can only be obtained if the corresponding assets have previously been locked on the source chain (i.e., ② ⇒ ①), and second, that the locked assets can only be released once the associated wrapped tokens have been burned (i.e., ④ ⇒ ③). The liveness property complements this by requiring eventual progress: if assets are locked on the source chain, the bridge must eventually mint the corresponding wrapped tokens on the destination chain (i.e., ① ⇒ ②); likewise, if wrapped tokens are burned on the destination chain, the bridge must eventually release the originally locked assets on the source chain (i.e., ③ ⇒ ④). Throughout, we consistently use the terms "source chain" and "destination chain" to denote the two blockchains involved.

A *trustless* bridge guarantees both safety and liveness without relying on any trust assumptions beyond those inherent to the source and destination chains. In other words, no external operator or committee needs to be trusted; correctness, safety, and liveness derive solely from chain consensus and verifiable on-chain evidence. An illustrative example is the recently proposed zkBridge [22]. In zkBridge, assuming that both the source and destination chains support EVM-style smart contracts SC_s and SC_d, respectively, the lock-and-mint paradigm is realized as follows. First, the user invokes SC_s to lock assets in the contract (step ①). Once this invocation is confirmed on the source chain, anyone–including the user–can obtain a proof of inclusion and submit it to SC_d to mint the corresponding wrapped tokens (step ②), ensuring the process remains trustless. Next steps mirror this approach: after the user burns the wrapped tokens in SC_d (step ③), a proof of inclusion on the destination chain is submitted to SC_s (step ④), which then releases the original assets back to the user.

A trustless bridge *fundamentally* relies on the property that every interaction with the source and destination chains produces an on-chain record that can serve as a proof of inclusion on the other chain. This mechanism ensures that all operations can be independently verified and executed without requiring trust in any intermediary. Moreover, a trustless bridge preserves *compatibility* with existing chains, facilitating practical deployment, as it does not require any hard fork or modifications to the underlying chains. However, the reliance on on-chain records fails for layer-2 payment channels such as the Lightning Network (LN) [1], where counterparties update shared state off-chain and only occasionally commit

to the base layer. As a result, standard trustless bridge designs cannot directly attest to channel state on a destination chain such as Ethereum.

Problem Description. The problem we address is enabling a trustless transfer of LN channel outcomes to a smart-contract chain (e.g., Ethereum) without introducing new trust assumptions or requiring changes to either LN or the destination chain, while preserving safety and liveness.

Motivating Illustrative Application. A bridge between the LN and Ethereum unlocks a new class of DeFi applications by turning *non-LN funds* into *temporary LN spending power*. Consider the following concrete payment scenario: Alice must pay Carol over LN (Carol only accepts Bitcoin/LN). Alice does have sufficient funds on Ethereum, but in the LN she has an open channel to Bob with only 5 coins of *outbound* capacity from Alice to Bob. Bob, in turn, has an LN channel to Carol with sufficient capacity to forward the payment. Without additional outbound capacity on the Alice–Bob channel, Alice can route at most 5 coins to Carol via the path Alice→Bob→Carol.

The bridge allows Alice to collateralize her Ethereum funds to temporarily increase her LN capacity on the Alice-Bob channel. First, Alice locks collateral worth α on Ethereum in a smart contract (step ①) that releases it back to Alice upon on-chain evidence that Bob has been repaid in LN. Next, Bob extends α coins of *temporary credit* to Alice by updating their Alice–Bob channel (step ②). This increases Alice's outbound capacity to $5+\alpha$, enabling her to route the LN payment to Carol through Bob (step ③).

Bob is compensated via an Ethereum fee, LN routing fees, or both. Once the forwarded payment ensures that Alice has repaid at least α to Bob within LN, the resulting channel state serves as the evidence for the Ethereum contract to release Alice's collateral (step ④). This example demonstrates how LN–Ethereum bridges can enhance liquidity and enable new cross-chain DeFi applications.

Technical Challenges. Designing a trustless, compatible bridge between LN and one chain (e.g., Ethereum) while maintaining safety and liveness is challenging. Consider two failure cases. First, after Alice locks collateral in the Ethereum smart contract (step ①), Bob might refuse to cooperate in their LN channel and decline to issue the α coins (step ②). This stalls progress and breaks the liveness guarantee. Second, suppose Alice does repay Bob in LN, so that the channel advances to a newer state s_n (step ③). She could then use s_n as proof to unlock her collateral in Ethereum (step ④) while simultaneously attempting to close the LN channel at an older state s_{n-1}, in which she still retains the α coins. This would allow Alice to reclaim both the collateral and the loan, violating safety.

These scenarios highlight the core challenge: without careful design, either party can exploit the mismatch between Ethereum's on-chain guarantees and LN's off-chain state updates to break safety or liveness.

Goal & Contributions. The objective of this work is to *design a trustless and compatible bridge between LN and Ethereum that guarantees both safety and liveness*. To achieve this goal, this work makes the following contributions:

- In Section 5, we contribute OptiBridge, a novel bridge between LN and Ethereum that is trustless and compatible with LN and Ethereum as deployed today. Moreover, OptiBridge ensures safety and liveness.
- In Section 6, we evaluate the performance of OptiBridge and compare it with the state-of-the-art Alba (NDSS'25) [17]. In the optimistic case where both users behave honestly, OptiBridge reduces the on-chain footprint by $\sim 73\%$ (from 4.51M gas in Alba to 1.22M gas in OptiBridge), resulting in significant cost savings for honest users. This is achieved by simplifying the optimistic path and selectively deploying the dispute resolution contract only when needed. In the pessimistic case where a dispute arises, OptiBridge still outperforms Alba, with a lower deployment cost and cheaper core dispute call (196K gas vs. 516K gas). We further show compatibility with Ethereum and LN specifications, whereas Alba necessitates modifications to standard LN transactions.
- In Section 7, we analyze the incentives of participants in OptiBridge and observe that the optimistic finalization path in OptiBridge is the Nash Equilibrium for both users. This implies that both users are incentivized to behave honestly, rather than resorting to the dispute resolution mechanism. Our approach in OptiBridge thereby resembles the optimistic paradigm that has been successfully deployed in practice in other blockchain settings, such as atomic swaps or rollups.

2 Background & Related Work

2.1 Smart Contracts in Ethereum

An Ethereum smart contract is EVM bytecode stored at an address and executed upon receiving a transaction. A deployment transaction carries the contract's Init function, which runs once and outputs the *runtime code* persisted at the created address. We denote a contract by SC, and a function call f with arguments $\boldsymbol{x}$ as $SC.\mathsf{f}(\boldsymbol{x})$. Each call executes atomically: either all state changes are applied or none at all. The call's input parameters are recorded on-chain as calldata, and if the code specifies event emissions, the EVM produces *logs* stored in the transaction receipt. These events act as lightweight signals that broadcast contract state changes to external observers, enabling efficient off-chain tracking.

Beyond executing functions, contracts can deploy new contracts at runtime using the `CREATE` opcode, which assigns a fresh address sequentially, or `CREATE2`, which deterministically computes an address from the deployer's address, a salt, and the contract's init code hash, allowing participants to precompute the address before deployment provided the same parameters are used. This property can be leveraged for *selective deployment* of a secondary contract. Selective deployment is achieved by committing to the `CREATE2`-derived address of a secondary contract. Upon receiving the agreed parameters, the main contract verifies the address against the commitment and, if consistent, deploys the contract, ensuring only the intended logic can be instantiated.

2.2 Payment Channel in the Lightning Network

A payment channel is a two-party protocol that enables multiple off-chain payments using only two on-chain transactions, one to open and one to close, thereby improving scalability for throughput-limited cryptocurrencies. The Lightning Network (LN) is the most widely deployed Bitcoin payment-channel system. We briefly overview the relevant mechanics and refer to [1,8] for further details.

Every user holds a digital-signature key pair (sk, vk) and is identified by their public key. Let Alice and Bob be two users in the Lightning Network. Alice holds (sk_A, vk_A) and Bob holds (sk_B, vk_B). Alice and Bob can operate a payment channel in three phases as follows.

In the first phase, Alice and Bob *open* a channel by publishing a funding transaction that locks γ_A and γ_B coins into a 2-of-2 output (multisig) controlled by (vk_A, vk_B). Once confirmed, the channel is opened with total balance $\gamma := \gamma_A + \gamma_B$. Spending the funding output requires signatures from both parties.

In the second phase, the parties *pay each other off-chain* by mutually authorizing new channel states that redistribute γ back to their individual addresses. For instance, the first transaction $tx := [(vk_A, \gamma_A - x), (vk_B, \gamma_B + x)]$ (or $tx := [vk_A \xrightarrow{x} vk_B]$ for simplicity) denotes the re-distribution of the initial γ coins locked in the channel so that Alice pays x coins to Bob. In general, each coin distribution in the channel (i.e., channel state i) is represented by a corresponding transaction that we denote by tx_i. The i-th channel state is authorized when Alice and Bob exchange their digital signatures $\sigma_{A,i}, \sigma_{B,i}$ for the corresponding transaction tx_i. After they do so, we say that Alice and Bob have agreed on the i-th channel state.

Finally, either party can *close* the channel by broadcasting any previously authorized tx_j, which settles coins on-chain according to the j-th state s_j.

A key subtlety is ensuring that, even after many states have been authorized, the channel is closed using the *latest* agreed state. LN achieves this with a punishment mechanism that (i) identifies who broadcast the closing transaction and (ii) revokes superseded states. For (i), each state i is represented by two commitment transactions, $tx_{A,i}$ and $tx_{B,i}$. Although these two transactions represent the same coin distribution, the concrete representation differs so that each transaction uniquely identifies the possible submitter of this transaction (e.g., by differing in how the vector $[(vk_A, \gamma_A + x), (vk_B, \gamma_B - x)]$ is sorted): for instance $tx_{A,i}$ identifies A as the submitter. Authorizing state i thus requires exchanging two signatures, one per commitment transaction in a given order. More concretely, assume that in the i-th state, Alice wishes to pay y coins to Bob. Since Alice is the payer, Alice first signs $tx_{B,i}[vk_A \xrightarrow{y} vk_B]$ to Bob so that Bob can get the y coins while identifying him as the submitter. Second, Bob signs $tx_{A,i}[vk_A \xrightarrow{y} vk_B]$, a transaction that also represents the i-th state, but identifies Alice as the submitter. For (ii), each transaction carries a commitment to a *revocation secret*. When state i is replaced by i+1, the parties exchange the revocation secrets for state i. If a revoked state later appears on-chain, the counterparty can use the revealed revocation secret to claim the cheater's funds during a penalty window, hence disincentivizing stale-state closes.

In summary, after funding the channel, the parties can perform arbitrarily many off-chain updates. The i-th state channel is updated as follows. First, they exchange the commitment to the revocation secret for the i-th state. Then, they create the transactions $tx_{A,i}$ and $tx_{B,i}$ representing the i-th state of the channel and subsequently exchange the pairs of signatures ($\sigma_{A,i}(tx_{B,i})$ and $\sigma_{B,i}(tx_{A,i})$) for $tx_{A,i}$ and $tx_{B,i}$ correspondingly. Note that Bob can produce $\sigma_{B,i}$ for $tx_{B,i}$ on their own, hence, the presence of $tx_{B,i}$ on-chain identifies Bob as the broadcaster (analogously for Alice and $tx_{A,i}$). Finally, Alice and Bob exchange the revocation secrets for the $i-1$-th state. This succinct description focuses on the details needed for our construction. We refer te reader to [1,8] for a complete description of the LN.

2.3 Related Work

Alba [17]. Alba is the first and only bridge system that permits transferring a LN state into a blockchain with expressive smart contract support, e.g., Ethereum. The approach in Alba to let an Ethereum contract validate an LN channel is to embed protocol-relevant information at the LN channel state itself. More concretely, Alba requires that channel states are defined with transactions of the form: $tx := (vk_A, vk_B) \rightarrow [(\gamma_A + x, vk_A), (\gamma_B - x, vk_B), (info)]$, where *info* denotes the protocol-relevant information (e.g., Alice returned the loan to Bob).[4] This information embedded in the channel state is therefore proof that both users have agreed on the bridge operation.

With this change in mind, after the *expected channel state* (e.g., the state where Alice returns the loan to Bob) is agreed upon by both participants in the LN, they hold (see Section 2.2): (i) the transactions $tx_{A,i}$ and $tx_{B,i}$ corresponding to the expected channel state i, each of them appended with the protocol-relevant *info* field; and (ii) the pairs of signatures $(\sigma_{A,i}(tx_{A,i}), \sigma_{B,i}(tx_{A,i}))$ and $(\sigma_{A,i}(tx_{B,i}), \sigma_{B,i}(tx_{B,i}))$ for $tx_{A,i}$ and $tx_{B,i}$ correspondingly.

This information permits the transfer of the channel state from the LN to the Ethereum contract. For that, Alice (or Bob) includes the following information in the Ethereum contract for the transfer to be effective: $\Pi := (tx_{A,i}, \sigma_{B,i}(tx_{A,i}), tx_{B,i}, \sigma_{A,i}(tx_{B,i}))$. Although it would still be correct to add it, note that $\sigma_{A,i}(tx_{B,i})$ is omitted because it is implicit that Alice agrees to their favorable representation of the channel state. Similar argument applies to omit $\sigma_{B,i}(tx_{B,i})$.

Moreover, Alba design requires that each state previous to the i-th state, where Alice returns the loan to Bob, is *locked* for a time in the future after the logic of the Ethereum contract has been correctly settled. This means that, although valid syntactically, LN states created in Alba can only be used after the Ethereum contract is settled, effectively opening the possibility to a grieffing attack from Alice if they decide not to cooperate with Bob.

The approach in Alba requires changing the transaction format used in the currently deployed LN implementations. For that, a new proposal for BOLT (the specification of the LN) must be proposed and analyzed by the community.

[4] Technically, this is possible using the OP_RETURN opcode available in Bitcoin.

Once approved, the different existing implementations of the LN need to be adapted accordingly. While theoretically plausible, it is a burden that would be interesting to avoid. OptiBridge instead permits to operate the LN channels as they are defined today, being therefore a fully backwards compatible solution with the current LN implementations.

Moreover, the Ethereum contract must include enough information to process Π, that is, the proof of the expected channel state. Effectively, this means that the Ethereum contract must store such a data structure, parse it, and verify its correctness. These operations are costly in terms of gas, which translates into monetary costs for the users. It is interesting to reduce this cost in a two-fold way: (i) reducing the information that needs to be stored in the smart contract; and (ii) consequently reducing the cost of parsing and verifying it. These are the goals of OptiBridge that are achieved in this work, as demonstrated with the evaluation in Section 6.

Chain to Chain Bridges. Several chain to chain bridges have been proposed so far, which we can roughly group into two categories. First, *SPV-based bridges* which rely on the concept of SPV clients. SPV light clients [16] verify block headers to identify the chain with the most Proof-of-Work and form the basis of chain relays [6]. Despite their theoretical appeal, relays see little deployment due to high costs ($\approx$ 100k gas per block header and 62k gas per transaction inclusion) and the absence of relaying incentives. Some systems, e.g., XClaim [25], still use SPV light clients with appropriate incentives. In contrast, OptiBridge requires no SPV verification: its contract merely parses a hash value and its pre-image in the optimistic finalization path, and a transaction and a signature pair in case of triggering a dispute.

Second, *zk-bridges*. Zero-knowledge (zk) interoperability solutions leverage zk protocols to ensure security. While zk-based relays have been proposed [21], only zkBridge [22] has gained traction. zkBridge, an EVM-compatible design, verifies state updates via zk-SNARKs and stores them in a stateful contract. However, it incurs high costs ($\approx$ 230k gas per proof verification, plus block storage) and lacks incentives to relay states among chains. Proof generation is also computationally intensive, requiring specialized hardware. In contrast, OptiBridge is consensus-agnostic: its proofs are lightweight, verifiable with standard lightweight cryptographic tools (e.g., hash functions and digital signatures), and feasible even in resource-constrained devices.

Cross-chain Cryptographic Protocols. A different line of work uses cryptographic primitives for cross-chain communication [5, 15, 23, 24] such as Hashed TimeLock Contracts (HTLCs) and adaptor signatures [4, 7, 19], primarily for atomic swaps [9, 12, 20]. These solutions trade expressiveness for simplicity yet still incur high costs ($\approx$ 270k gas per HTLC). Like OptiBridge, they are consensus-agnostic and impose a minimal on-chain footprint. However, their scope is limited to token swaps, whereas OptiBridge generalizes to DeFi protocols such as loans. OptiBridge, in fact, builds upon atomic swaps (see Section 3).

3 Solution Overview

Our approach minimizes on-chain storage and computation on Ethereum while remaining trustless and backward-compatible with today's Lightning Network. Fig. 1 summarizes a deployment of OptiBridge between the Lightning Network and Ethereum at a high level. In short, verifier (Bob) samples a secret r_{bridge}, commits to it with $h_{bridge} := \mathsf{H}(r_{bridge})$, and shares h_{bridge} with the prover (Alice). They then deploy the bridge contract $SC_{bridging}$, funded with Alice's collateral (step ①). The contract exposes two clauses: (i) if a submitted s^* satisfies $\mathsf{H}(s^*) = h_{bridge}$, release the collateral to the prover (Alice); (ii) otherwise, allow either party to enter a dispute phase instantiated by an optional, on-demand dispute contract $SC_{dispute}$. Either party can trigger dispute resolution if cooperation fails; the dispute contract is instantiated only when needed.

Contract initialization is coupled with the LN loan transfer (step ②) via a standard atomic swap protocol [20]. Atomicity of the atomic swap ensures that steps ① and ② either both succeed or both abort, yielding safety and liveness at setup. This mechanism is backward compatible with current LN implementations and Ethereum, and it introduces no additional trust assumptions, achieving trustlessness and compatibility. After setup, the channel proceeds per the LN spec. OptiBridge finalizes either *optimistically* when both parties cooperate, or via *dispute resolution* when a party deviates.

Optimistic Finalizing. When the channel reaches the repayment state s_i (step ③), Bob delivers r_{bridge} to Alice off-chain. Alice submits it on-chain to $SC_{bridging}$ to recover her collateral (step ④). The LN channel remains usable thereafter with no changes to the protocol.

Beyond this, when the optimistic finalizing is the common case, the dispute logic can be moved out of the main contract. A separate $SC_{dispute}$, identified by $h_{SC} := \mathsf{H}(SC_{dispute})$ and deployable via CREATE2, executes the dispute only when instantiated, avoiding baking complex logic into the primary bridge.

Dispute Resolution at a Glance. If cooperation fails, either party can instantiate $SC_{dispute}$ and present succinct evidence tied to the LN repayment state. The dispute flow covers early closes on stale states, withholding of r_{bridge}, and global timeouts; funds are awarded to the correct party after a bounded window, ensuring safety and liveness while externalizing pessimistic costs to rare cases.

Dispute Resolution Details. The dispute mechanism must protect honest users against a misbehaving counterparty. It covers deviations from the honest protocol and penalizes the at-fault party upon valid proof. Let s_i denote the channel state where Alice returns the loan and s_{i-1} the preceding state; honest progress is to reach s_{i-1} and then s_i. The dispute resolution considers two scenarios: before s_{i-1} is reached and after s_{i-1} is reached.

If Alice decides to close the channel before s_{i-1} is reached (e.g., to refuse paying back the loan), Bob can wait for `T_BRIDGE` to expire in the $SC_{dispute}$ contract and get compensated with Alice's collateral (case ⓐ). If Bob closes before s_{i-1} (e.g., blocking repayment in LN to seize collateral in Ethereum), Alice can present the corresponding channel state to $SC_{dispute}$ and recover the collateral (case ⓑ); both parties are therefore incentivized to reach s_{i-1}.

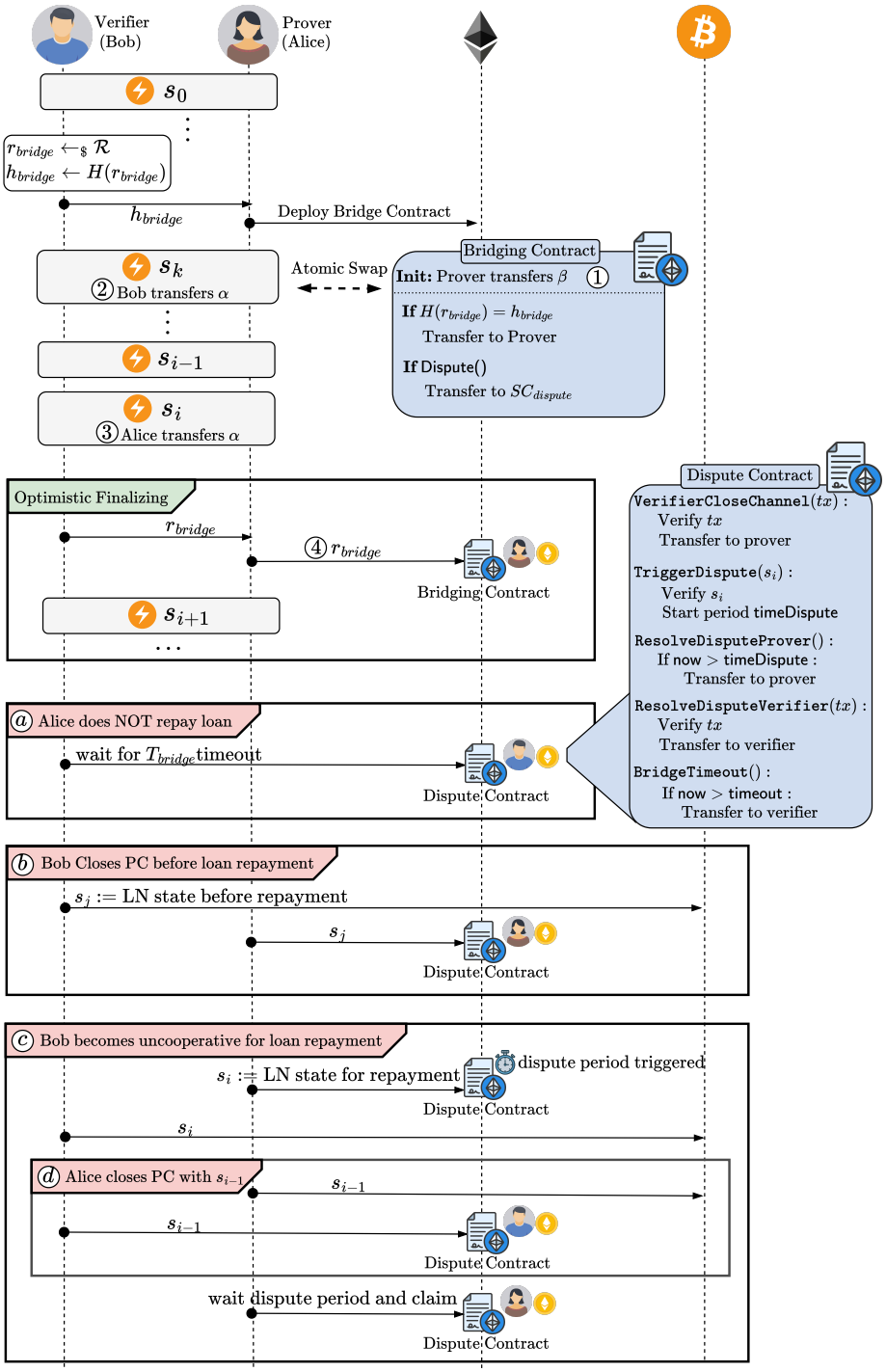

Fig. 1. High-level bridge workflow between Lightning and Ethereum in OptiBridge.

After s_{i-1} is reached, the dispute mechanism needs to handle cases where Bob does not cooperate with Alice to transition from s_{i-1} to s_i in Lightning, or does not share the secret r_{bridge}. For that, the dispute mechanism permits Alice to present the state s_i, effectively signaling Alice's will to continue with the protocol. Then Bob can read this state s_i from the contract and publish it in Bitcoin to effectively close the channel and get the loan back (case ⓒ).

There is a subtlety here. At this point, both LN states s_{i-1} and s_i are valid and not revoked yet: s_{i-1} has been validly reached through LN whereas s_i has been exposed to Bob through the Ethereum contract. This means that when Bob attempts to close the channel with state s_i, a malicious Alice can simultaneously attempt to close the channel with state s_{i-1}, which is more beneficial for Alice since the loan is not repaid at this state. The Bitcoin blockchain unequivocally resolves this race condition: only one of the two conflicting states will be recorded on the Bitcoin blockchain and effectively close the channel.

Putting this last case of the dispute together (case ⓒ): Alice first presents s_i to the dispute resolution contract to signal her willingness to return the loan to Bob. Then, suppose at any point while Bob is attempting to close the channel with s_i, Bob observes that Alice attempted to close the channel with s_{i-1} (e.g., by observing s_{i-1} in the mempool or on the Bitcoin blockchain after being accepted). In that case, Bob can present s_{i-1} to the dispute contract to signal misbehavior from Alice and get the collateral (inner case ⓓ). Only after a time window sufficient for the Bitcoin blockchain to include s_i, if Bob decides to do so, can Alice claim the collateral back at the Ethereum contract.

We analyze the incentives of the participants in OptiBridge (Section 7) and conclude that the optimistic finalization path is the preferred strategy for rational users. Moreover, this path preserves the safety and liveness guarantees of OptiBridge for steps ③ and ④, while introducing no additional trust assumptions and remaining fully compatible with existing LN and EVM-compatible blockchains such as Ethereum.

4 System Model and Goals

This section frames the environment in which OptiBridge operates and the guarantees it aims to provide. We outline the system and threat model for the source and destination ledgers, and describe the safety, liveness, trustlessness, and compatibility goals that guide our design.

4.1 System and Threat Model

We model the destination ledger $\mathcal{L}_D$ as a Lightning Network payment channel jointly operated by Alice (the prover) and Bob (the verifier), and we denote each off-chain update by the state transition Δ_{s_D} introduced in Section 2.2. As in the standard Lightning security model, we assume that neither participant will settle the channel using a *revoked state* because doing so triggers the well-known punishment transaction that assigns the entire balance to the

counterparty. We assume that the Lightning Network relies on Bitcoin as the underlying blockchain. The source chain $\mathcal{L}_S$ is an Ethereum-like smart-contract platform where a transaction tx induces the transition Δ_{s_S} by executing the smart contract.

Two mutually distrusting parties, Alice (prover) and Bob (verifier), control key pairs (sk_A, vk_A) and (sk_B, vk_B) in both ledgers. Both parties are rational: they deviate from the honest execution of the protocol only if doing so yields them higher utility.

System Assumptions. We assume that (a) both Ethereum and Bitcoin offer standard *consensus liveness* (valid transactions appear on-chain within a bounded delay, denoted δ_{BTC} and δ_{ETH}) and *consensus safety* (persistence of confirmed transactions). (b) We assume *observability*: any transaction broadcast on one chain is observable by an honest counterparty within the respective delay. (c) The system timeouts are *derived* from these synchrony bounds to prevent race conditions. Specifically, the dispute window `T_DISPUTE` must accommodate the full "round-trip" required to observe a fault and penalize it. We require $\texttt{T\_DISPUTE} > \delta_{BTC} + 2 \cdot \delta_{ETH} + \Delta$, where Δ is a safety margin. This ensures that even if an adversary times a Bitcoin state closure for the worst-case network congestion (δ_{BTC}), the honest party has sufficient time to generate a fraud proof and have it included on Ethereum (δ_{ETH}) before the window closes. For context, with conservative estimates (e.g., $\delta_{BTC} \approx 1$ hour, $\delta_{ETH} \approx 15$ mins), a parameter of $\texttt{T\_DISPUTE} = 24$ hours comfortably satisfies the security condition.

Model Scope & Robustness. We analyze the interaction as a game of perfect information between rational agents to formally derive the system's incentives (Section 7). We acknowledge that this abstraction simplifies real-world frictions such as mempool congestion, fee market volatility, and Miner Extractable Value (MEV). Crucially, however, OptiBridge *decouples* economic incentives from cryptographic safety. While the Nash Equilibrium relies on rationality, the protocol's safety and liveness do *not*. Even against irrational or "spiteful" adversaries who deviate from the equilibrium at their own cost, the bridge guarantees fund safety provided the synchrony assumptions (timeouts $\gg$ network delays) hold. Thus, the system degrades gracefully: failure of the rationality assumption incurs on-chain costs but does not result in theft.

Monitoring & Compatibility. This system model places a burden on honest users to monitor the source ledger and respond to adversarial actions within the specified time windows (e.g., Alice must watch Bitcoin for Bob's early close attempt). This requirement is identical to the standard security model of the Lightning Network. Consequently, OptiBridge is fully compatible with existing ecosystem solutions: users may delegate these monitoring tasks to third-party services (watchtowers) just as they do for standard Lightning channels. This ensures that the liveness requirement does not impose a new type of operational overhead on the user.

4.2 Security and System Goals

We follow the model in [17] to define the notion *bridge*. As in [17], we refer to states and state transitions borrowing the terminology of distributed computing to provide formal definitions.

Definition 1 (Bridge [17]). *A* bridge *is a tuple of algorithms* (Γ_S, Γ_D) *defined as follows:*

- $\Gamma_S(\mathcal{L}_D, \Delta_{s_D})$ *is the algorithm running on* $\mathcal{L}_S$ *that, on input* $\mathcal{L}_D$ *and a state transition* Δ_{s_D} *of* $\mathcal{L}_D$*, outputs a state transition* Δ_{s_S} *to be applied to* $\mathcal{L}_S$*.*
- $\Gamma_D(\mathcal{L}_S, \Delta_{s_S})$ *be the algorithm running on* $\mathcal{L}_D$ *that, on input* $\mathcal{L}_S$ *and a state transition* Δ_{s_S} *of* $\mathcal{L}_S$*, outputs a state transition* Δ_{s_D} *to be applied to* $\mathcal{L}_D$*.*

We say that a bridge is correct *if for every valid state transition* Δ_{s_D} *in* $\mathcal{L}_D$ *and* Δ_{s_S} *in* $\mathcal{L}_S$*,* $\Gamma_S(\mathcal{L}_D, \Delta_{s_D})$ *outputs a* valid *state transition* Δ_{s_S} *to be applied to* $\mathcal{L}_S$ *and* $\Gamma_D(\mathcal{L}_S, \Delta_{s_S})$ *outputs a* valid *state transition* Δ_{s_D} *to be applied to* $\mathcal{L}_D$*.*

Security Goals. We next define the notions of safety and liveness for a bridge.

Definition 2 (Bridge Safety). *A bridge is* safe *if the following holds:*

- *Let* $\Delta_{s_S} \leftarrow \Gamma_S(\mathcal{L}_D, \Delta_{s_D})$*. If* Δ_{s_S} *is applied to* $\mathcal{L}_S$*, then* Δ_{s_D} *must be in* $\mathcal{L}_D$*.*
- *Let* $\Delta_{s_D} \leftarrow \Gamma_D(\mathcal{L}_S, \Delta_{s_S})$*. If* Δ_{s_D} *is applied to* $\mathcal{L}_D$*, then* Δ_{s_S} *must be in* $\mathcal{L}_S$*.*

Definition 3 (Bridge Liveness). *A bridge is* live *if the following holds:*

- *Let* $\Delta_{s_S} \leftarrow \Gamma_S(\mathcal{L}_D, \Delta_{s_D})$*. If* Δ_{s_D} *is in* $\mathcal{L}_D$*, then* Δ_{s_S} *can be applied to* $\mathcal{L}_S$*.*
- *Let* $\Delta_{s_D} \leftarrow \Gamma_D(\mathcal{L}_S, \Delta_{s_S})$*. If* Δ_{s_S} *is in* $\mathcal{L}_S$*, then* Δ_{s_D} *can be applied to* $\mathcal{L}_D$*.*

System Goals. We finally describe the goals of trustlessness and compatibility.

A bridge achieves *trustlessness* if it does not require any trust assumption additional to those already in $\mathcal{L}_S$ and $\mathcal{L}_D$. A bridge achieves *compatibility* if it can be seamlessly deployed with existing implementations of both $\mathcal{L}_S$ and $\mathcal{L}_D$.

5 OptiBridge: Protocol Details

We now detail the protocol that achieves the goals in Section 4. It comprises the Ethereum on-chain contracts, a LN state-update interface, and an off-chain flow that coordinates the optimistic and dispute paths. The design minimizes common-case cost while preserving a secure fallback if cooperation fails.

5.1 Source Chain (Ethereum) Contract

Figure 2 summarizes the bridge logic deployed on $\mathcal{L}_S$. The on-chain main contract $SC_{bridging}$ exposes three entry points: (i) Init records the parties, the collateral and loan amounts, the commitment to the optimistic finalization secret h_{bridge}, and the commitment to the dispute contract $h_{dispute}$; (ii) OptimisticProof

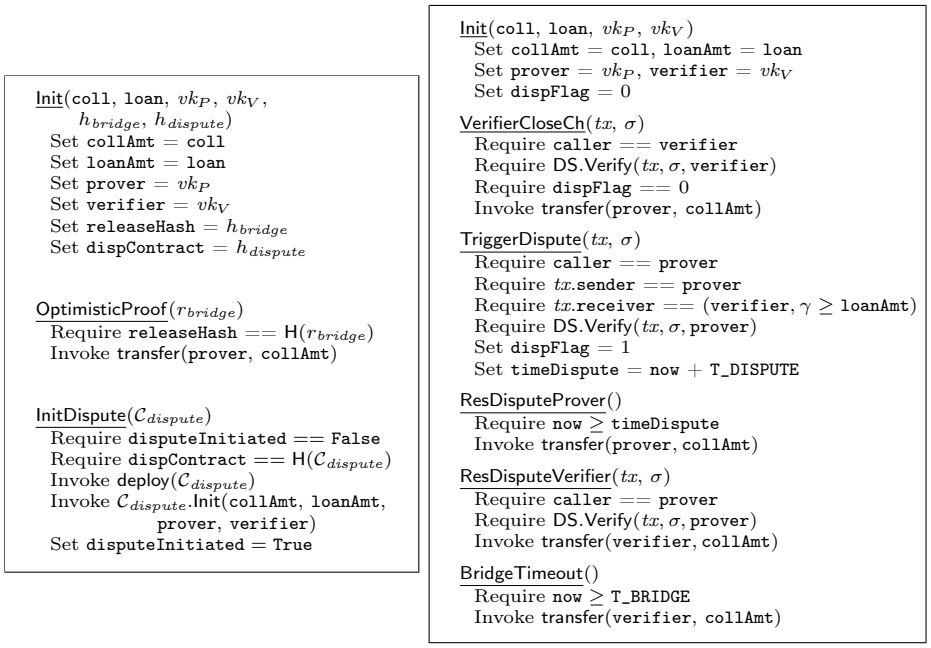

Fig. 2. $SC_{bridging}$ contract (left) and $SC_{dispute}$ contract (right). Here, `T_DISPUTE` denotes a time window large enough to safely process the closing of an LN channel.

lets Alice recover the collateral as soon as she reveals a preimage that matches h_{bridge}; and (iii) InitDispute instantiates the dispute contract only when cooperation fails, ensuring that the optimistic path never pays for the pessimistic code. Once deployed, the dispute contract $SC_{dispute}$ enforces the pessimistic flow. First, VerifierCloseCh allows Alice to reclaim the collateral if Bob prematurely publishes a Lightning state that precedes the repayment; the flag `dispFlag` prevents this shortcut once the formal dispute has started. Second, TriggerDispute records Alice's repayment evidence, sets `dispFlag`, and opens a timer `T_DISPUTE` during which Bob can answer. Inside that window, Bob may call ResDisputeVerifier to prove that Alice closed the channel with an earlier state, whereas Alice waits until the timer expires and then calls ResDisputeProver to recover the funds. Finally, BridgeTimeout lets Bob collect the collateral if the global deadline `T_BRIDGE` elapses without a successful repayment.

5.2 Lightning States Updates

As recalled in Section 2.2, OptiBridge follows the Lightning state-update protocol verbatim and we introduce the notation used later in this section. For each party $\hat{P} \in \{P, V\}$ and state index k, let $r_{\hat{P},k}$ denote the revocation secret

Setup Phase (vk_P, vk_V, `collAmt`, `loanAmt`, $\mathcal{C}_{dispute}$, `T_BRIDGE`)

1. V computes $r_{bridge} \leftarrow \{0,1\}^\lambda$, $h_{bridge} \leftarrow \mathsf{H}(r_{bridge})$, and sends h_{bridge} to P.
2. Using a standard atomic swap protocol, P and V atomically execute the following two actions:
   (a) P computes $h_{dispute} \leftarrow \mathsf{H}(\mathcal{C}_{dispute})$ and invokes $SC_{bridging}.\mathsf{Init}(\texttt{collAmt}, \texttt{loanAmt}, vk_P, vk_V, h_{bridge}, \texttt{T\_BRIDGE}, h_{dispute})$, computes $r_{P,0} \leftarrow \{0,1\}^\lambda$, $hr_{P,0} \leftarrow \mathsf{H}(r_{P,0})$ and sends $hr_{P,0}$ to V.
   (b) V computes P's channel state $tx_{P,0}[V \xrightarrow{\texttt{collAmt}} P]$, $\sigma_{V,0} \leftarrow \mathsf{Sign}(sk_V, tx_{P,0})$, $r_{V,0} \leftarrow \{0,1\}^\lambda$, $hr_{V,0} \leftarrow \mathsf{H}(r_{V,0})$, and sends $(\sigma_{V,0}, hr_{V,0})$ to P.
3. P computes V's channel state $tx_{V,0}[V \xrightarrow{\texttt{collAmt}} P]$, $\sigma_{V,0} \leftarrow \mathsf{Sign}(sk_V, tx_{P,0})$, $r_{P,0} \leftarrow \{0,1\}^\lambda$, $hr_{P,0} \leftarrow \mathsf{H}(r_{P,0})$, and sends $(\sigma_{P,0}, hr_{P,0})$ to V.

Channel Updates

For each state update $k \in [1,n]$, with value γ_k, sender $S \in \{P,V\}$ and receiver $R \in \{P,V\}$ update their LN channel as described in Section 2.2.

Loan Return

Let i be the channel state where P wishes to finish returning the loan to V in LN.

1. P computes $r_{P,i} \leftarrow \{0,1\}^\lambda$, $hr_{P,i} \leftarrow \mathsf{H}(r_{P,i})$ and sends $hr_{P,i}$ to V. Wait for V to answer, otherwise *go to Handle Dispute*.
2. V computes $r_{V,i} \leftarrow \{0,1\}^\lambda$, $hr_{V,i} \leftarrow \mathsf{H}(r_{V,i})$ and sends $hr_{V,i}$ to P.
3. P computes V's channel state $tx_{V,i}[P \xrightarrow{\texttt{loanAmt}} V]$, $\sigma_{P,i} \leftarrow \mathsf{Sign}(sk_P, tx_{V,i})$, and sends $\sigma_{P,i}$ to V. Wait for V to answer, otherwise *go to Handle Dispute*.
4. V computes P's channel state $tx_{P,i}[P \xrightarrow{\texttt{loanAmt}} V]$, $\sigma_{V,i} \leftarrow \mathsf{Sign}(sk_V, tx_{V,i})$, and sends $\sigma_{V,i}$ to P.
5. P revokes P's $i-1$ channel state by sending $r_{P,i-1}$ to V. Wait for V to answer, otherwise *go to Handle Dispute*.
6. V accepts loan return and revokes V's $i-1$ channel state by sending $(r_{bridge}, r_{V,i-1})$ to P.
7. P invokes the contract $SC_{bridging}.\mathsf{OptimisticProof}(r_{bridge})$.

Fig. 3. OptiBridge Protocol (Part I).

and $hr_{\hat{P},k}$ its committed hash. Moreover, $tx_{\hat{P},k}[\hat{P} \xrightarrow{x} \hat{Q}]$ denotes the transaction representing the k-th state, with $\hat{P}$ as submitter and where $\hat{P}$ pays x coins to $\hat{Q}$.

5.3 The Off-chain Protocol in OptiBridge

This section specifies the off-chain interaction between the prover (P) and the verifier (V) with the Lightning Network and the Ethereum contracts to implement secure bridging. The full message flow appears in Figs. 3 and 4.

Setup Phase. V sends P the parameters needed to initialize $SC_{bridging}$: the commitment to the loan-finalization secret h_{bridge} and the commitment to the dispute-contract code $h_{dispute}$. P and V run an atomic-swap style procedure that (i) initializes $SC_{bridging}$ with P's collateral and the commitments $(h_{bridge}, h_{dispute})$,

Handle Dispute

Let i be the channel state where P wishes to finish returning the loan to V in LN.

1. P checks if $SC_{dispute}$ contract is deployed, otherwise P invokes $SC_{bridging}$.InitDispute($\mathcal{C}_{dispute}$). Then, P invokes $SC_{dispute}$.TriggerDispute($tx_{V,i}$, $\sigma_{P,i}$).
2. V reads $tx_{V,i}$ and $\sigma_{P,i}(tx_{V,i})$ from $SC_{dispute}$ contract, computes $\sigma_{V,i} \leftarrow \mathsf{Sign}(sk_V, tx_{V,i})$ and tries to close the LN channel with the tuple ($tx_{V,i}$, $\sigma_{P,i}(tx_{V,i})$, $\sigma_{V,i}(tx_{V,i})$).
3. If P closed the LN channel in state $i-1$, V retrieves the tuple ($tx_{P,i-1}$, $\sigma_{P,i-1}$, $\sigma_{V,i-1}$) from the source ledger and invokes $SC_{dispute}$.ResDisputeVerifier($tx_{P,i-1}$, $\sigma_{P,i-1}$).
4. If P does not close the channel with state $i-1$, P waits for `T_DISPUTE` to expire and invokes $SC_{dispute}$.ResDisputeProver().

Monitor Source Ledger

Let j be the channel state published by V before the loan is returned.

1. P reads the tuple ($tx_{V,j}$, $\sigma_{V,j}(tx_{V,j})$, $\sigma_{P,j}(tx_{V,j})$) from the source ledger. Check if $SC_{dispute}$ is deployed, otherwise invoke $SC_{bridging}$.InitDispute($\mathcal{C}_{dispute}$). Invoke $SC_{dispute}$.VerifierCloseCh($tx_{V,j}$, $\sigma_{V,j}$).

Handle Timeout

Assume that `T_BRIDGE` bridge has expired in the destination ledger.

1. V checks if $SC_{dispute}$ is deployed, otherwise invokes $SC_{bridging}$.InitDispute($\mathcal{C}_{dispute}$). V invokes $SC_{dispute}$.BridgeTimeout().

Fig. 4. OptiBridge Protocol (Part II).

and (ii) advances their LN channel to the lending state where V extends the loan to P. After setup, the channel proceeds with standard LN updates.

Loan Return. P attempts to repay the loan to V on LN. If V fails to respond at any step, P switches to dispute. The expected exchange is: (1) P sends V the commitment to the revocation secret for the target state s_i (the repay state); (2) V replies with the corresponding revocation secret; (3) P authorizes s_i and sends it to V; (4) V returns its authorization for s_i; (5) P revokes s_{i-1} to V; (6) V reciprocates (revokes s_{i-1}) and provides the final secret r_{bridge}; (7) P calls $SC_{bridging}$.OptimisticProof(r_{bridge}) to reclaim the collateral.

Handle Dispute. If P initiates a dispute, P deploys $SC_{dispute}$ and triggers it with evidence for s_i (including P's authorization). (2) Using s_i, V closes the LN channel at s_i to recover the loan on Bitcoin. (3) If P maliciously broadcasts s_{i-1}, V presents the proof of misbehavior and claims the collateral via $SC_{dispute}$. (4) Otherwise, once s_i is confirmed on Bitcoin, P claims the collateral via $SC_{dispute}$.

Monitor Source Ledger. P runs a *scheduler* that monitors Bitcoin for any unilateral LN close by V outside the dispute flow. If detected, P submits the observed state to $SC_{dispute}$ to claim the collateral.

Handle Timeout. V runs a *scheduler* that monitors the bridge timeout `T_BRIDGE`. If `T_BRIDGE` expires, V claims the collateral via $SC_{dispute}$.

6 System Evaluation

This section quantifies the gas cost of running OptiBridge on Ethereum in both the optimistic and the pessimistic paths. We explain how much gas users spend when everything runs smoothly, how much they pay when a dispute is raised, and how these figures compare with the Alba bridge [17].

Experimental environment. We benchmark gas usage with Hardhat network 2.17.4, deploying a fresh instance of each Solidity contract for every call. Our implementation of the Bitcoin transaction verification logic leverages the open-source libraries from the Alba protocol[5]. The harness replays the calldata assembled in Section 5, so each signature, timeout, and hash mirrors the protocol flow. We compile with the same optimizer configuration used in the implementation and average three executions per entry (standard deviation $< 0.1\%$), reporting the resulting mean.

Measurement scope. We profile two execution tracks. The optimistic track covers the steady-state flow where the prover reveals r_{bridge} and no dispute contract is deployed. The pessimistic track exercises the full escalation: the prover or verifier triggers the dispute mechanism, and the settlement logic selects the winning player. For each call we record the raw gas returned by `eth_estimateGas`. Alba's numbers come from their public proof-of-concept and Table II, which evaluate the same payback scenario; additional application logic would add its own marginal cost, but we keep the focus on the bridge core.

Gas reference. Table 1 lists the measured gas consumption for OptiBridge alongside the published Alba values. The upper block covers the optimistic operations, the middle block focuses on dispute-specific calls, and the lower block shows one-time deployment overheads. Entries marked with "—" indicate that the function does not exist in the corresponding system. All figures report the mean over three repetitions; the negligible variance did not warrant additional aggregation.

Optimistic cost breakdown. The optimistic flow comprises two calls. The bridge setup (428,939 gas) initialises the contract, stores both public keys, registers the hash of the repayment secret, and caches the dispute template, which explains the modest overhead versus Alba's 393,401 gas deployment. The subsequent 'optimisticProof' (40,107 gas) simply checks the preimage of h_{bridge} and releases the escrow. Alba expends 48,027 gas for the same step because it validates two signatures on a sequence number, and it requires an additional 'SubmitProof' transaction (253,566 gas) to transmit the full channel state—work that OptiBridge avoids by keeping the proof to a single hash opening.

Pessimistic cases and dispute flow. When cooperation fails we observe three escalation paths, each reflected in Table 1. In all cases, if the dispute

[5] Specifically, the DS.Verify logic referenced in Fig. 2 utilizes the contracts available at `https://github.com/ALBA-blockchain/ALBA-Protocol`.

Table 1. Gas usage comparison between OptiBridge and Alba

Function	OptiBridge	Alba
setup (bridge)	428,939	393,401
optimisticProof	40,107	48,027
submitProof	—	253,566
bridgeTimeout	36,971	—
setup (dispute)	428,939	—
verifierCloseChannel	183,993	—
dispute	196,438	515,860
resolveDispute	175,688	168,046
settle	30,279	49,814
bridge contract deployment	1,215,315	4,513,248
dispute contract deployment	2,785,514	—

contract is not already on-chain, the bridge first deploys and initialises it via `setup (dispute)`; the bullets below list only the branch-specific calls:

(i) **Deadline expiry.** Once `T_BRIDGE` elapses, the verifier calls `timeoutBridge` (36,971 gas) to reclaim the collateral.

(ii) **Verifier closes early.** If the verifier publishes a stale Lightning state on Bitcoin, Alice invokes `verifierCloseChannel` (183,993 gas) with the signed transaction retrieved from the Bitcoin chain. Beyond the regular transaction validity checks (i.e., one HTLC output for the verifier and one P2PKH for the prover), the contract verifies that the transaction is signed by both parties, thereby proving the verifier's malicious behavior[6]. The call then immediately transfers the escrow to Alice.

(iii) **Verifier withholds r_{bridge}.** The prover initiates a dispute by calling `dispute` (196,438 gas) and submitting the signed state s_i that reflects loan repayment. Similar to `verifierCloseChannel`, this function verifies that the repayment amounts are correct, the outputs are well formed, and the prover's signature is valid. It also opens a dispute window.
During this time, although the prover submits s_i on Ethereum, it could still attempt to close the Lightning channel with an older state s_{i-1}. The dispute window prevents such behavior: the verifier can counter by invoking `resolveDispute` (175,688 gas), which performs the same checks as in the function `verifierCloseChannel` and transfers the collateral to the verifier. If the prover is honest, it simply waits for the dispute window to expire and then calls `settle` (30,279 gas) to move the funds to its account.

Side-by-side comparison with Alba. Table 1 highlights two points. First, once the bridge is deployed, the optimistic path is cheaper in OptiBridge: our

[6] For the verification of the lightning (Bitcoin) transactions, DS.Verify in Fig. 2, we have used the Alba contracts which can be accessed at https://github.com/ALBA-blockchain/ALBA-Protocol

steady-state proof is 40,107 gas vs. Alba's 48,027 gas, and we avoid Alba's 253,566-gas `SubmitProof` entirely. Second, Alba keeps dispute deployment cheap by embedding dispute logic in a monolithic contract, but this makes every user pay for Lightning parsing even when nothing goes wrong. In contrast, OptiBridge pays the 3.4 M-gas dispute deployment *only when a dispute occurs*; day-to-day operation never touches the dispute code, matching our goal to optimize the common case and isolate pessimistic work on demand.

Qualitatively, Alba alters LN by requiring per-commitment `OP_RETURN` payloads and relative timelocks that grow with each update, enlarging Bitcoin transactions and forcing Ethereum to validate extra fields during proofs and disputes. OptiBridge leaves LN unchanged: no extra timelocks, custom outputs, or `OP_RETURN`, so off-chain messages remain small and Ethereum checks reduce to hash preimages and standard signatures. This simplifies dispute verification and explains the lower optimistic gas footprint: honest parties pay only for a single hash opening, while rare disputes externalize their cost.

System Goals Achieved. Our measurements show that OptiBridge meets the objectives in Section 4. It is *trustless*: all actions rely solely on publicly verifiable on-chain evidence (standard signatures and hash preimages), with no relays, oracles, operators, or committees. It is *compatible*: it does not require any LN protocol changes, nor custom outputs or `OP_RETURN`. Moreover, it only relies on ordinary contract calls on Ethereum.

7 Security and Incentive Analysis

In this section, we first analyze the incentives of the rational players, prover and verifier. We show that at the equilibrium, rational players will opt for the optimistic finalization path of the bridge. Then, we show that the optimistic finalization path in OptiBridge provides both safety and liveness.

7.1 Incentive Analysis

We analyze the prover-verifier interaction in OptiBridge as an extensive form game with perfect information, where players make sequential moves as depicted in Fig. 5. Following the backward induction technique in [10], we identify the subgame perfect Nash Equilibrium (SPNE) [18]. This analysis is consistent with recent blockchain protocol analysis [3, 13], including the state-of-the-art Alba [17].

Game Specification. Our game specification models the interaction between two rational players, prover P and verifier V. The game tree captures the multi-stage dynamics in OptiBridge with decisions points for:

- **Initial stage:** V decides whether to collaborate with P to reach the LN channel state where P pays back the loan (LN state s_i), or close the LN channel before (with state $s_{j<i}$), with the consequent dispute resolution in favor of P as depicted in ⓑ.

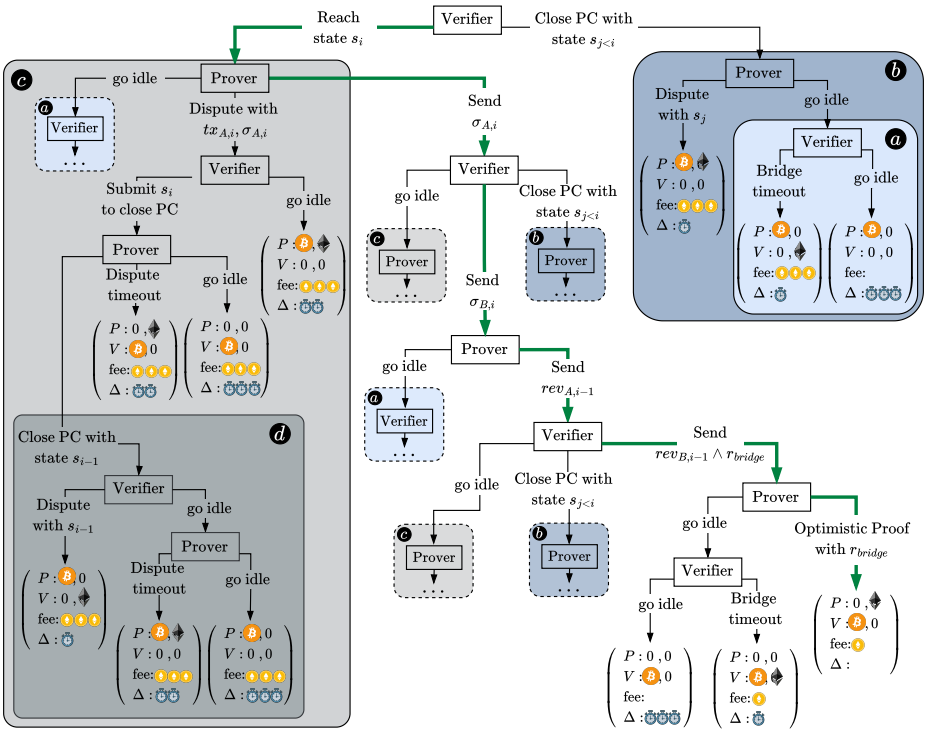

Fig. 5. The game between prover and verifier in OptiBridge. The green path depicts the ideal execution (optimistic path), where both users follow the honest protocol. For readability, colored dotted boxes depict collapsed summaries of the detailed boxes shown in the solid boxes of the same color. *fee* denotes the on-chain fees required to reach the payout (less fees is better), and Δ denotes the number of time windows that users need to wait to reach the payout (less is better). A time window is either a transaction being included in Bitcoin, in Ethereum, or the timeout for the overall bridge operation.

- **Loan Return:** P decides whether to follow the optimistic path to return the loan (i.e., move the LN channel to state s_i), or do it through the dispute resolution path, as depicted in ⓒ. Therefore, this opens the subsequent two subtrees: *optimistic path*, and *dispute resolution*.
- **Optimistic Path:** Starting with P, P and V take turns to honestly exchange the information required for moving the LN channel to state s_i, that is, $\sigma_{A,i}(tx_{B,i})$, $\sigma_{B,i}(tx_{A,i})$, $r_{A,i-1}$ and the pair $(r_{B,i-1}, r_{bridge})$. Finally, P invokes $SC_{bridging}$.OptimisticProof on input r_{bridge}. For each of these steps, P and V can decide to deviate by not answering (i.e., go idle). If any of P or V goes idle, the counterparty moves to the corresponding subgame ⓐ, ⓑ, or ⓒ in the dispute resolution.
- **Dispute Initialization:** In ⓒ, P initializes the dispute resolution by presenting the pair $(tx_{B,i}, \sigma_{A,i})$, corresponding to the LN state s_i, to the $SC_{dispute}$

contract. V can use s_i to close the LN channel, and the game enters into two possible subgames, as described next.

- **Resolution from P:** This subgame models the option where, after V closed the channel with state s_i, P honestly resolves the dispute by waiting for the dispute window expiration and claiming the collateral. Alternatively, P could opt to race V in closing the channel with state s_{i-1}, opening the door to the final subgame of this analysis.
- **Resolution from V:** In (d), if P closed the channel with state s_{i-1}, V can simply solve the dispute on their favor presenting s_{i-1} as proof of P's misbehavior, or refuse to resolve dispute. In the latter, P can finish the last step by claiming back the collateral after the dispute window expires (or simply go idle as the default option for all moves in this game).

Equilibrium Analysis. Now, we apply the standard concepts of strategy profiles and subgame perfect Nash equilibrium [10, 18] to our OptiBridge analysis. More concretely, for each player P and V we define their strategy profile actions across all stages. For instance, P's strategy is defined by actions "Dispute with $(tx_{A,i}, \sigma_{A,i})$", "go idle", "Send $\sigma_{A,i}$" and so on. Similarly, we can define V's strategy. We employ backward induction [10] to identify the unique existing equilibrium where rational players prefer the honest, optimistic execution of OptiBridge to avoid the unnecessary and costly on-chain dispute resolution.

Theorem 1 (OptiBridge Equilibrium). *Consider the game between P and V derived from the tree show in Fig. 5. Let players be rational with the objective of maximizing the utility. Let the payout on Ethereum be the complete collateral (denoted by ETH symbol in Fig. 5), the payout on Bitcoin be the complete loan amount (denoted by BTC symbol in Fig. 5), the fee be the level of fees that users incur to execute the corresponding payout, and the Δ be the time delay that users need to wait to perform an action. P and V prefer to obtain ETH and BTC over 0, and keep the fees and time delays as low as possible. Finally, let $\Sigma^* := (\sigma_P^*, \sigma_V^*)$ be the strategy profile for P and V correspondingly, where:*

- σ_P^**: (Send* $\sigma_{A,i}$*, Send* $r_{A,i-1}$*, invoke* $SC_{bridging}$.OptimisticProof(r_{bridge})*).*
- σ_V^**: (Reach state* s_i*, Send* $\sigma_{B,i}$*, Send* $(r_{B,i-1}, r_{bridge})$*)*

Then, Σ^* constitutes the unique Subgame Perfect Nash Equilibrium (SPNE).

The SPNE strategy in Theorem 1 achieves (i) *incentive compatibility*: deviations result in possible loan and collateral loss for one of the parties, as well as higher fees and/or longer time delays; and (ii) *efficiency*: the equilibrium path incurs only off-chain communication between the parties, except for a single setup and simple claim of the collateral in the Ethereum contract. We defer the proof of Theorem 1 to Section A.

7.2 Security Analysis

Theorem 1 establishes that the optimistic finalization path is the equilibrium for rational players in OptiBridge. Given that, we now discuss that the optimistic finalization path in OptiBridge is both safe and live.

Proof (OptiBridge is safe). To demonstrate that OptiBridge is safe, we must establish that two conditions are met:

- Let $\Delta_{s_S} \leftarrow \Gamma_S(\mathcal{L}_D, \Delta_{s_D})$. If Δ_{s_S} is applied to $\mathcal{L}_S$, then Δ_{s_D} must be in $\mathcal{L}_D$.
- Let $\Delta_{s_D} \leftarrow \Gamma_D(\mathcal{L}_S, \Delta_{s_S})$. If Δ_{s_D} is applied to $\mathcal{L}_D$, then Δ_{s_S} must be in $\mathcal{L}_S$.

First case. Here Δ_{s_D} is the state where the prover pays the loan back to the verifier in the LN channel, whereas Δ_{s_S} is the state where the prover obtains the collateral back from the Ethereum contract. This case holds because the prover (Alice) can only get the collateral back from the Ethereum contract if she obtained the secret r_{bridge} from Bob in LN, which implies that Bob must have received the loan back in LN previously.

Second case. Here, Δ_{s_D} is the state where the verifier lends the loan coins to the prover in LN, whereas the Δ_{s_S} is the state where the prover locks the collateral in Ethereum, as part of the initialization of $SC_{bridging}$. This case holds straightforwardly from the atomicity property of the atomic swap protocol used to couple the two state transitions.

Proof (OptiBridge is live). To demonstrate that OptiBridge is live, we must establish that two conditions are met:

- Let $\Delta_{s_S} \leftarrow \Gamma_S(\mathcal{L}_D, \Delta_{s_D})$. If Δ_{s_D} is in $\mathcal{L}_D$, then Δ_{s_S} can be applied to $\mathcal{L}_S$.
- Let $\Delta_{s_D} \leftarrow \Gamma_D(\mathcal{L}_S, \Delta_{s_S})$. If Δ_{s_S} is in $\mathcal{L}_S$, then Δ_{s_D} can be applied to $\mathcal{L}_D$.

First case. Here Δ_{s_D} is the state where the prover pays the loan back to the verifier in the LN channel, whereas Δ_{s_S} is the state where the prover obtains the collateral back from the Ethereum contract. This case holds because if the prover (Alice) has paid the loan back to Bob, she has received the optimistic secret r_{bridge} that permits the prover to obtain the collateral back in Ethereum *on her own*, without the further help of the verifier. Therefore, the prover on their own can make the transition Δ_{s_S} happen in $\mathcal{L}_S$ (i.e., claim the collateral back).

Second case. Here, Δ_{s_D} is the state where the verifier lends the loan coins to the prover in LN, whereas the Δ_{s_S} is the state where the prover locks the collateral in Ethereum, as part of the initialization of $SC_{bridging}$. This case holds straightforwardly from the atomicity property of the atomic swap protocol used to couple the two state transitions.

8 Conclusions

This work introduces OptiBridge, a trustless, cost-efficient bridge between the LN and Ethereum that remains fully compatible with both systems as they are currently deployed. OptiBridge finalizes optimistically via a single hash-preimage check and invokes a richer dispute contract only on demand, preserving safety and liveness without external operators or changes to LN.

Our measurements show markedly lower on-chain costs that state-of-the-art: optimistic deployment is 1.22M vs. 4.51M gas ($\sim$ 73% lower) and proof submission is 40,107 vs. 253,566 gas; when disputes arise, we deploy the dispute contract then (2,785,514 gas) and the core dispute call is cheaper (196,438 vs. 515,860). Incentive analysis identifies the optimistic path as a Nash equilibrium, and the system achieves safety, liveness, trustlessness, and compatibility.

Future work includes exploring optimizations such as replacing preimage checks in $SC_{bridging}$ with adaptor signatures [4, 7] and leveraging programmable payment channels [11, 14] to migrate Ethereum contract logic off-chain.

Acknowledgments. This work was supported by the grant CEX2024-001471-M and the grant PID2022-142290OB-I00, both funded by MICIU/AEI/10.13039/501100011033, FEDER, UE.

References

1. The bitcoin lightning network: Scalable off-chain instant payments, https://lightning.network/lightning-network-paper.pdf
2. Coinmarketcap website, https://coinmarketcap.com
3. Aumayr, L., Avarikioti, Z., Maffei, M., Mazumdar, S.: Securing lightning channels against rational miners. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 393–407. ACM (2024). https://doi.org/10.1145/3658644.3670373, https://doi.org/10.1145/3658644.3670373
4. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized channels from limited blockchain scripts and adaptor signatures. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT. Lecture Notes in Computer Science, vol. 13091, pp. 635–664. Springer (2021). https://doi.org/10.1007/978-3-030-92075-3_22
5. Christodorescu, M., English, E., Gu, W.C., Kreissman, D., Kumaresan, R., Minaei, M., Raghuraman, S., Sheffield, C., Wijeyekoon, A., Zamani, M.: Universal payment channels: An interoperability platform for digital currencies (2021), https://arxiv.org/abs/2109.12194
6. Frauenthaler, P., Sigwart, M., Spanring, C., Sober, M., Schulte, S.: Eth relay: A cost-efficient relay for ethereum-based blockchains. In: Proceedings of the 2020 IEEE International Conference on Blockchain (Blockchain 2020). pp. 204–213. IEEE (2020). https://doi.org/10.1109/Blockchain50366.2020.00032, https://ieeexplore.ieee.org/document/9495004
7. Gerhart, P., Schröder, D., Soni, P., Thyagarajan, S.A.K.: Foundations of adaptor signatures. In: Joye, M., Leander, G. (eds.) Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14652, pp. 161–189. Springer (2024). https://doi.org/10.1007/978-3-031-58723-8_6, https://doi.org/10.1007/978-3-031-58723-8_6

8. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: Bonneau, J., Heninger, N. (eds.) Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 12059, pp. 201–226. Springer (2020). https://doi.org/10.1007/978-3-030-51280-4_12, https://doi.org/10.1007/978-3-030-51280-4_12
9. Hoenisch, P., Mazumdar, S., Moreno-Sanchez, P., Ruj, S.: Lightswap: An atomic swap does not require timeouts at both blockchains. In: García-Alfaro, J., Navarro-Arribas, G., Dragoni, N. (eds.) Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2022 International Workshops, DPM 2022 and CBT 2022, Copenhagen, Denmark, September 26-30, 2022, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13619, pp. 219–235. Springer (2022). https://doi.org/10.1007/978-3-031-25734-6_14, https://doi.org/10.1007/978-3-031-25734-6_14
10. Kuhn, H.: 6. Extensive Games and the Problem of Information. Contributions to the Theory of Games II (1953) 193-216., pp. 46–68. Princeton University Press, Princeton (1997). https://doi.org/doi:10.1515/9781400829156-011, https://doi.org/10.1515/9781400829156-011
11. Kumaresan, R., Le, D.V., Minaei, M., Raghuraman, S., Yang, Y., Zamani, M.: Programmable payment channels. In: Pöpper, C., Batina, L. (eds.) Applied Cryptography and Network Security - 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part III. Lecture Notes in Computer Science, vol. 14585, pp. 51–73. Springer (2024). https://doi.org/10.1007/978-3-031-54776-8_3, https://doi.org/10.1007/978-3-031-54776-8_3
12. Malavolta, G., Moreno-Sánchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS 2019). Internet Society (2019), https://www.ndss-symposium.org/ndss-paper/anonymous-multi-hop-locks-for-blockchain-scalability-and-interoperability/
13. Minaei, M., Kumaresan, R., Beams, A., Moreno-Sanchez, P., Yang, Y., Raghuraman, S., Chatzigiannis, P., Zamani, M., Le, D.V.: Scalable off-chain auctions. In: Proceedings of the 33rd Annual Network and Distributed System Security Symposium (NDSS 2026). Internet Society (2026)
14. Minaei Bidgoli, M.M., Kumaresan, R., Yang, Y., Raghuraman, S., Zamani, M., Christodorescu, M., Gu, W.: Universal payment channel system and method. US Patent Application US20250045751A1 (2025), https://patents.google.com/patent/US20250045751A1/en, published Feb. 6, 2025
15. Minaei Bidgoli, M.M., Kumaresan, R., Zamani, M., Gaddam, S.: Universal payment channels. US Patent US11995623B2 (2024), https://patents.google.com/patent/US11995623B2/en, issued May 28, 2024
16. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
17. Scaffino, G., Aumayr, L., Bastankhah, M., Avarikioti, Z., Maffei, M.: Alba: The dawn of scalable bridges for blockchains. In: 32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025. The Internet Society (2025), https://www.ndss-symposium.org/ndss-paper/alba-the-dawn-of-scalable-bridges-for-blockchains/

18. Selten, R.: Reexamination of the perfectness concept for equilibrium points in extensive games. International Journal of Game Theory **4**(1), 25–55 (1975). https://doi.org/10.1007/bf01766400
19. Tairi, E., Moreno-Sanchez, P., Schneidewind, C.: Ledgerlocks: A security framework for blockchain protocols based on adaptor signatures. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023. pp. 859–873. ACM (2023). https://doi.org/10.1145/3576915.3623149, https://doi.org/10.1145/3576915.3623149
20. Thyagarajan, S.A., Malavolta, G., Moreno-Sánchez, P.: Universal atomic swaps: Secure exchange of coins across all blockchains. In: Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP). pp. 1299–1316. IEEE (2022). https://doi.org/10.1109/SP46214.2022.9833731, https://ieeexplore.ieee.org/document/9833731
21. Westerkamp, M., Eberhardt, J.: zkrelay: Facilitating sidechains using zksnark-based chain-relays. In: Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 378–386. IEEE (2020). https://doi.org/10.1109/EuroSPW51314.2020.00063, https://ieeexplore.ieee.org/document/9094274
22. Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., Song, D.: zkbridge: Trustless cross-chain bridges made practical. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022. pp. 3003–3017. ACM (2022). https://doi.org/10.1145/3548606.3560652, https://doi.org/10.1145/3548606.3560652
23. Zamani, M., English, E., Kumaresan, R., Christodorescu, M., Gu, W., Harper, C., Sheffield, C., Minaei, M., Raghuraman, S.: Cross-border payments for central bank digital currencies via universal payment channels. Retrieved from Bank for International Settlement website: http://www. bis. org/events/cpmi_ptfop/proceedings/paper14. pdf. b (2021)
24. Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sanchez, P., Kiayias, A., Knottenbelt, W.J.: Sok: Communication across distributed ledgers. In: Borisov, N., Díaz, C. (eds.) Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 12675, pp. 3–36. Springer (2021). https://doi.org/10.1007/978-3-662-64331-0_1, https://doi.org/10.1007/978-3-662-64331-0_1
25. Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., Knottenbelt, W.J.: Xclaim: Trustless, interoperable, cryptocurrency-backed assets. In: Proceedings of the 40th IEEE Symposium on Security and Privacy (IEEE S&P 2019). pp. 193–210. IEEE (2019). https://doi.org/10.1109/SP.2019.00028, https://ieeexplore.ieee.org/document/8835224

A Extended Incentive Analysis

In this section, we include the proof for Theorem 1.

Proof. We argue by backward induction on the game in Fig. 5, using the standard Lightning assumption that no rational party settles the channel with a revoked state (publication triggers punishment and a strict loss).
Terminal subgames. Payoffs at terminal nodes are determined by the deployed contracts. Along the optimistic path, P reclaims the collateral on Ethereum and V obtains repayment on LN, incurring minimal fees and delay. Any path that enters dispute yields weakly higher fees and weakly longer delay for the deviator, and never improves the ultimate payout relative to the optimistic outcome.
Optimistic micro-steps. At any optimistic decision node (exchanging signatures on s_i, revealing revocations for s_{i-1}, releasing r_{bridge}), the mover chooses Continue or Deviate. Deviating (idling or attempting to settle at a prior state) transitions to a dispute subgame where the counterparty has a best response that enforces the intended outcome (closing at s_i or letting the dispute timer expire) while the deviator bears extra fees/delay. By the LN punishment assumption, settling with a revoked state is strictly dominated. Hence, Continue strictly dominates Deviate at every optimistic micro-step. At the final optimistic node, after s_i is mutually authorized and r_{bridge} is available, P's unique best action is to call $SC_{bridging}.\mathsf{OptimisticProof}(r_{bridge})$; withholding only increases delay and risks collateral loss.
Off-ramps to dispute. (i) If V closes the LN channel before s_i, P's best response is to initialize dispute, after which P recovers the collateral (and V cannot improve upon the intended outcome). Thus pre-emptive closing is strictly worse for V than cooperating. (ii) If P initiates dispute instead of continuing optimistically, V's best response is to finalize s_i on LN, after which P still recovers the collateral but now with additional fees/delay; thus P is strictly worse than staying optimistic. (iii) If P initiates dispute and V closes with s_i, P's best response is to wait out the dispute window and reclaim the collateral; racing to close at s_{i-1} only adds fees/delay and potential loss of collateral. This scenario introduces a new subgame: if P closes at s_{i-1}, V's best response is to resolve the dispute in its favor, which recovers the loan amount and collateral while imposing extra fees/delay on P. If V does not resolve, P reclaims the collateral after the dispute window expires. In either case, P is strictly worse off than following the optimistic path.

By backward induction, at every decision node the unique best response keeps playing on the optimistic path, culminating in P invoking $\mathsf{OptimisticProof}$. Therefore, Σ^* is the unique SPNE.