2025 IEEE 45th International Conference on Distributed Computing Systems (ICDCS) | 979-8-3315-1723-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICDCS63083.2025.00105

Tombolo: Towards a Decentralized Interconnected Blockchain Ecosystem using Cross-Chain Payment Channels

Siddharth Maurya*, Nitin Awathare†, Vinay Joseph Ribeiro*, Umesh Bellur*
*Department of Computer Science, Indian Institute of Technology Bombay
savvysiddharth@gmail.com, {vinayr, umesh}@cse.iitb.ac.in
†Department of Computer Science, Indian Institute of Technology Jodhpur
nitina@iitj.ac.in

***Abstract*—The world of finance and decentralized applications has undergone a revolution with the advent of blockchain technology, resulting in the emergence of various blockchain platforms. This underscores the necessity for conducting exchanges between different blockchains to facilitate blockchain interoperability. However, the challenges linked to the scalability of individual blockchains pose negative implications during inter-chain exchanges. A prominent solution to tackle the scalability challenge in a blockchain, particularly in terms of throughput, is a Payment Channel Network. However, creating such a solution for facilitating inter-blockchain exchange presents significant challenges, as it demands atomic state updating on two different blockchains. In this paper, we introduce a novel, fully distributed mechanism for establishing cross-chain payment channels (CCPC), designated as Tombolo protocol, designed to operate seamlessly across different blockchains. Furthermore, we provided the sequence of steps that the user should follow to facilitate further exchange without contacting either of the involved blockchains. Additionally, we showcase its resilience against malicious behaviour, specifically in situations where a participant attempts to close the channel using an outdated state or becomes inactive during exchanges. Furthermore, we have demonstrated the viability of Tombolo by implementing it on an Ethereum Virtual Machine-based blockchain using Solidity v0.8.0 based smart contracts. Through comprehensive experimentation and evaluation, we illustrate that CCPC can be established with approximately 2.5 times the gas utilization and around twice the time overhead compared to Raiden, a state-of-the-art intra-chain channel.**

***Index Terms*—payment channel, blockchain interoperability, cross-chain bridge, scalability**

I. Introduction

As the number of blockchain platforms has increased, each with its unique features and use cases, the need for seamless interoperability between these disparate networks has becomes increasingly crucial. The interoperability problem arises due to the isolated nature of individual blockchain networks. Each blockchain operates on its own set of rules, consensus mechanisms, and smart contract languages, resulting in data silos and hindering the exchange of value and information across different chains. This lack of communication and coordination between blockchain networks presents significant challenges for the efficient and frictionless transfer of assets, limiting the full potential of blockchain technology.

Various protocols [1]–[6] have been proposed to enable the transfer of assets between different blockchain networks. However, their effectiveness is constrained by the pace of the slowest blockchain. For instance, if two users—one on *Blockchain A* network with an average block time of 10 minutes and one on *Blockchain B* network with 15 seconds of average block time—seek to perform an exchange, it will take at least a duration of 10 minutes. This is due to the 10-minute block inter-arrival time of former Blockchain A, regardless of the fact that latter Blockchain B has a 15-second block inter-arrival time.

The challenge of extended block times, leading to prolonged transaction latency and potentially reduced transaction throughput in a blockchain, is addressed through the introduction of the *Payment Channel*. Hitherto, we had payment channels which operate with accounts within the same blockchain. Known intra-chain payment channels include the Lightning Network [7] for Bitcoin [8] and the Raiden Network [9] for Ethereum [10]. Though the implementations of these payment channels may differ based on their respective platforms, the fundamental mechanism remains the same. A payment channel between two parties is initialized by performing a Funding transaction, which locks certain amount of tokens from both participants, acting as proof of the initial balance of the payment channel. Once the channel is established, participants can conduct an unlimited number of off-chain transactions instantaneously, without requiring network confirmations. The channel balance (i.e., the funds held by both parties) is updated off-chain as transactions occur, reflecting the distribution of funds between the participants. When either party chooses to close the channel, the final state is broadcasted to the main blockchain, settling the last balance between them, and the funds return to their respective wallets based on the latest channel state.

Inspired by the success of Payment Channels within a single blockchain, where both users share the same blockchain, this paper introduces a cross-chain payment channel (CCPC)

2575-8411/25/$31.00 ©2025 IEEE
DOI 10.1109/ICDCS63083.2025.00105

protocol called *Tombolo*[1]. This protocol effectively addresses the aforementioned performance limitations caused by the slowest blockchain in the context of cross-chain exchanges. However, implementing decentralized cross-chain payment channels poses significant challenges, particularly in ensuring consistency across multiple blockchains. Tombolo provides a well-defined sequence of transactions for the formation and termination of cross-chain payment channels to ensure the channel state remains consistent across both involved blockchains. Furthermore, Tombolo proposes mechanisms to address any potential inconsistencies in the channel state between the blockchains due to malicious or offline participants, thereby enhancing its reliability and robustness. For on-chain verification of transactions from the other blockchain, it utilizes relay nodes to bring blockchain headers from another blockchain network and verify the existence of the target transaction in those other blocks using Merkle Proof [12]. Essentially, it operates as a light client [13] for an external blockchain network within a smart contract, thereby facilitating the capability to verify transactions from a remote blockchain.

It is important to note that Tombolo is agnostic to the specific mechanism used for verifying such transactions. In this paper, for the design and implementation of Tombolo, we adopt the aforementioned methodology to enable on-chain verification. Moreover, Tombolo can be integrated with intra-chain payment channel systems, enabling a cross-chain bridge between two separate payment channel networks.

To ensure the viability of Tombolo, we have implemented it under the name *Cross Network Payment Channel System (XNPCS)* on Ethereum Virtual Machine (EVM) based blockchains. XNPCS comprises two smart contracts: XPC, facilitating cross-chain transfers, and SPC, enabling the establishment of intra-chain payment channels. In our experiments involving two different blockchain networks, we demonstrated that the creation of SPC incurs a gas cost similar to the gas for channel creation in Raiden, a state-of-the-art protocol used in this work. Additionally, we illustrated that creating an XPC requires gas usage approximately 2.5 times and twice the overhead time of creating a channel in the Raiden Network, irrespective of the block inter-arrival time, which is just a one-time overhead. Furthermore, we observed that channel creation and closure times, along with their various components, exhibit uniform growth with block inter-arrival time.

In summary, we make the following contributions in this paper:

- We introduce Tombolo, which is a fully distributed approach to establishing payment channels across two distinct blockchain networks. This is achieved by proposing an organized set of actions that each participant in the channel should follow.
- We have demonstrated the resilience of Tombolo against malicious behavior, where either participant deviates from the protocol.
- We offer a tangible implementation of Tombolo and validate its feasibility through a comparative analysis with Raiden, a state-of-the-art protocol.

The remainder of the paper unfolds as follows: Section II delves into the related work. A comprehensive explanation of Tombolo protocol is presented in Section III. Section V showcases the implementation and experimental evaluation of Tombolo. Finally, Section VI concludes the paper with a discussion.

II. Related Work

To achieve seamless interoperability between diverse blockchain networks, various solutions address cross-chain asset transfer challenges. This section reviews existing approaches, including extensions of intra-chain payment channels for cross-chain transfers [3], [4], [14], relayed block headers [5], [6], [15]–[17], Atomic Swaps [2], Wrapped Bitcoin [18], and RenBTC [19].

Cross-chain transfers in the Lightning Network [3] connect Bitcoin's payment channel network with similar networks on other blockchains by relying on a participant with funds in both networks, who acts as a relay between them. This approach enables interoperability without creating new cross-chain payment channels. A similar technique is used in Cross-Chain Virtual Payment Channels [4]. However, this requires participants to distribute funds across blockchains, reducing liquidity—an issue cross-chain payment channels aim to address. CrossChannel [14] achieves secure and scalable cross-chain micropayments through a chain relay mechanism that employs a Proof-of-Stake-based consensus among relayers to ensure correctness. While this approach provides robust security, it introduces additional complexity and operational overhead. In contrast, our protocol's implementation utilizes ETH Relay [15], which simplifies the relayer's role by requiring them to relay only block headers, eliminating the need for a consensus-based mechanism among relayers. Furthermore, our protocol achieves lower gas costs during channel creation, as detailed in the experimentation section.

BTC Relay [5] enables Bitcoin transaction verification on Ethereum through a decentralized smart contract system that uses relay nodes to submit Bitcoin block headers. This setup allows smart contracts to verify Bitcoin transactions without a trusted third party. However, BTC Relay is limited by its one-way transfer capability and the need for on-chain confirmations. Cross-chain payment channels offer a more efficient alternative, providing instant, low-cost transactions without on-chain confirmations.

Atomic cross-chain swaps [2] facilitate direct, trustless asset exchanges across blockchains using hashed time-locked contracts. These swaps execute atomically, ensuring both transactions complete together or cancel entirely. While they enable secure, decentralized interoperability, atomic swaps may face higher failure rates and increased latency due to mandatory on-chain confirmations, unlike faster payment channels.

XClaim [16] serves as an alternative to Atomic Swaps by enabling cross-chain transfers through smart contracts that

[1]*Tombolo:* a sand or gravel bar connecting an island with the mainland or another island [11]

store sender blockchain headers. This process locks tokens on the sender's blockchain and generates equivalent tokens on the receiver's blockchain, validated via proof of burn using block headers alone. Though XClaim suggests extending its approach for cross-chain payment channels, it does not provide a specific protocol to achieve this.

Wrapped Bitcoin (wBTC) [18] bridges BTC to Ethereum by minting wBTC tokens via a central custodian, who receives BTC and issues wBTC to the owner's Ethereum address. While useful, this model introduces security risks due to custodial centralization [20], contrasting with decentralized token transfer norms. RenBTC addresses this by decentralizing custody through the Ren protocol, which uses Shamir's Secret Sharing [21] and Secure Multiparty Computation [22]. However, both require on-chain confirmation for each transfer.

zkBridge [6] proposes a trustless cross-chain bridge using zk-SNARKs for zero-knowledge proof validation of block headers from the source chain. While on-chain validation is efficient, off-chain proof generation can be costly and slow, affecting cross-chain transaction latency. Unlike cross-chain payment channels, zkBridge requires zero-knowledge proofs of block headers for each transfer. Though zk-SNARKs could validate blocks in our protocol, we have not evaluated their compatibility in this work.

LayerZero [17] provides a protocol for cross-chain messaging by verifying transaction inclusion with block headers, leveraging Chainlink [23], [24] as an oracle. Although Chainlink is deemed decentralized, its network is controlled by the Chainlink team [20], [25], raising trust concerns. LayerZero also relies on relay nodes for proof relaying, assuming independence between oracles and relay nodes—introducing potential security risks if they collude.

III. Cross-Chain Payment Channel Protocol

In this section, we commence by outlining the assumptions made regarding the blockchain networks participating in the exchange. Subsequently, we delve into the crucial components and flow of the protocol, encompassing channel formation, off-chain transfers, and channel termination. Additionally, we provide an in-depth exploration of various failure scenarios and the resilience of the proposed protocols against them.

A. Assumptions

For the proper operation of CCPC formed with the Tombolo protocol, we have established certain assumptions about the blockchain networks engaged in the exchange. Firstly, we assume that the blockchain network must offer support for Turing-complete smart contracts. These smart contracts play a crucial role in verifying merkle proofs related to the token locking process on the counterpart blockchain. Additionally, we presume the existence of relay nodes that facilitate the forwarding of block headers from one blockchain to the other, adopting a similar approach to BTC Relay. Moreover, we posit that each participant engaged in CCPC possesses an address in each blockchain involved in the channel. This assumption is reasonable, as the majority of blockchains allow tokens to be transferred within addresses residing on the same blockchain and do not permit addresses outside the blockchain to introduce new native tokens. However, the introduction of native tokens can occur through consensus, typically in the form of mining rewards. Furthermore, we leverage this assumption to introduce the novel channel state, the details of which are elaborated in section III-B.

Considering the aforementioned assumptions, our subsequent discussion will focus on channel creation, transfers within the established channel, and the termination of the channel.

B. Channel State Schema

Before delving into channel creation, let's briefly discuss the channel state, representing the contributions of the parties involved in the channel. The naive channel is created between two users residing on the same blockchain network, referred to as an *intra-chain* payment channel. Each such channel is created using a funding transaction on the blockchain that locks the funds from both parties. Upon channel creation, the parties can transact with each other off-chain instantly and without fees and keep a log of the latest balance. Consider users α and β participating in the channel, where they initially contribute amounts x and y to the channel ch, respectively. We refer to the users involved in the channel as participants. Therefore, throughout this paper, we will use these terms, i.e., user and participant, interchangeably. Given this, the state S of the channel is defined as $S = \{\alpha : x, \beta : y\}$. A transaction of amount z from α to β (with the constraint that z does not exceed α's current balance) updates the channel state from $S = \{\alpha : x, \beta : y\}$ to $S = \{\alpha : x - z, \beta : y + z\}$. The updated state indicates the remaining contribution, which we will refer to as the *channel balance* of α and β is $x - z$ and $y + z$ respectively. Likewise, β has the capability to transfer any amount to α, within the constraints of its current balance in the channel. Subsequently, the state is updated to reflect these changes.

The representation of the naive channel state is insufficient for capturing the complexities of the CCPC state. Therefore, in the Tombolo protocol we introduce a novel channel state schema comprising four balances, each representing the balance of an individual party on the two blockchains. This differs from the existing intra-chain payment channel, like the Lightning Network [7], which has only two balances.

For example, let's consider two users, α and β, where α owns tokens on Blockchain B_1 and β owns tokens on Blockchain B_2, and they wish to create a channel ch. Furthermore, α makes an initial contribution of x_1 to the channel from the tokens he owns on B_1. Similarly, β makes an initial contribution of y_2 to the channel from the tokens he owns on B_2. In the illustrated scenario, as presented in state 0 of Figure 1, the CCPC state S is represented as $S = \{\alpha : (x_1, y_1),\ \beta : (x_2, y_2)\}$. Here, y_1 and x_2 are initialized to 0, symbolizing the initial contributions of α and β to the channel using tokens from B_2 and B_1 respectively. To elaborate on Tombolo in this paper, we are utilizing the same

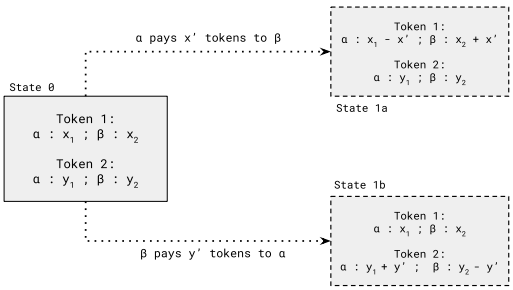

Fig. 1: CCPC State and state transition post token transfer in Tombolo protocol

scenario, where α possesses tokens on B_1 and β holds tokens on B_2. As a result, we will interchangeably use α's blockchain and B_1, as well as β's blockchain and B_2 throughout the paper. Now, having defined the payment channel state, let's delve into the channel creation next.

C. Channel Creation

In this section, we have outlined the steps for creating the CCPC. To enhance comprehension, let's first walk through the process of creating a naive channel.

1) Channel Creation in naive State Channel: Opening a channel necessitates participants to deposit funds to a smart contract on the main chain, thereby initializing the channel state. This is accomplished through the execution of a main chain transaction known as the *funding transaction*. The funding transaction essentially involves invoking the payable function within the smart contract.

The deposited funds also function as a bond, ensuring the commitment of participants to honest behavior. In the event of malicious behavior during dispute resolution, the contract deducts the deposit of the guilty parties as a penalty. Participants in the channel must collectively endorse an initial state that serves as the genesis of the state channel, allowing users to commence transactions thereafter.

2) Channel Creation in Tombolo: In order to establish a payment channel between two blockchain networks, a primary step is setting up a funding transaction, which includes initial balances. However, this presents a unique challenge, as the state updated by such a transaction needs to be synchronized between both blockchains. Simply creating a single funding transaction in each blockchain independently would not suffice, as one blockchain network would not be aware of the state of the other network's account.

To address the challenges mentioned above, we divide the channel creation phase into two sub-phases, each governed by a transaction. These transactions correspond to the components of the funding transaction, namely *Propose Funding* transaction, and *Confirm Funding* transaction.

To initiate a CCPC, both participants must first agree on their contribution in the channel, off-chain, after which they will broadcast a *Propose Funding* transaction on their respective blockchain. User who creates the funding transaction

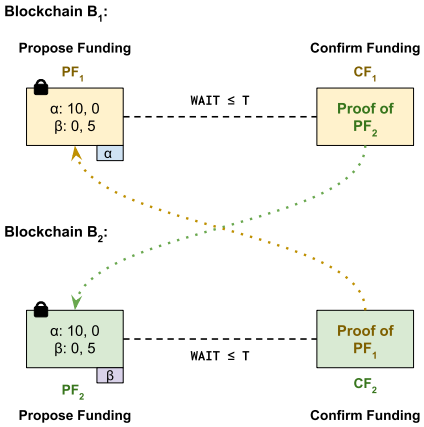

Fig. 2: Funding Phase when both participants could produce the Propose Funding proof from their counter-part Blockchain

should sign it before they broadcast it. For example, as depicted in Figure 2, user α and β agrees on their contribution of 10 and 5 on Blockchain B_1 and B_2, respectively. Furthermore, α initiates a funding transaction, PF_1, specifying his individual contribution of 10 on B_1 and 0 on B_2. PF_1 also incorporates the initial contribution of β of 0 in B_1 and 5, an off-chain agreed value, on B_2. Similarly, β generates PF_2 on blockchain B_2. These *Propose Funding* transactions serve as the initial *Balance Proof* of the payment channel. Moreover, α and β sign PF_1 and PF_2, respectively. These funding transactions lock the mentioned funds on the respective blockchains for a minimum time period, 'T', which can be defined in terms of the number of blocks.

However, PF_1 and PF_2 alone are insufficient for establishing the payment channel due to a lack of trust across the blockchain network. For instance, β might decide to lock a different amount of funds than the off-chain agreed amount, choosing to lock 4 while the off-chain agreed fund is 5. As a result, PF_1 would indicate individual contributions as $\alpha : (10, 0)$, $\beta : (0, 5)$, whereas PF_2 would show $\alpha : (10, 0)$, $\beta : (0, 4)$. Additionally, β could intentionally abstain from committing PF_2.

To avoid the above-mentioned issues and to establish trust, both participants need to present a Merkle Proof to their respective blockchain networks, demonstrating that their partner's balance is locked in the respective *Propose Funding* transaction. In other words, α needs to provide a proof to B_1 that PF_2 is committed to blockchain B_2, while β should provide proof to B_2 that PF_1 is committed to blockchain B_1. Participants can either communicate the proof to each other or obtain it from their partner's blockchain. Once they possess a valid Merkle proof, they include it in the *Confirm Funding* transaction and broadcast this transaction to their respective blockchain networks. For instance, α includes the proof for PF_2 in CF_1 and broadcasts it to B_1, as illustrated in Figure 2. It's crucial that the corresponding *Confirm Funding* transactions are confirmed on the respective blockchains within the

specified time T, concluding the channel creation phase with the initial state of $S = \{\alpha : (10, 0),\ \beta : (0, 5)\}$, as illustrated in our example in Figure 2.

3) Handling Failure: In the scenario where one participant fails to commit a *Confirm Funding* transaction while the other participant successfully does so, the participant who successfully commits their *Confirm Funding* transaction might risk having their funds locked indefinitely. To mitigate this risk, the Tombolo protocol allows for the unlocking of funds after a timeout period T. To initiate the fund unlocking process, someone must submit a *Funding Discarded (FD)* transaction on the blockchain where the *Confirm Funding* transaction has timed out. It's crucial to note that the *Funding Discarded (FD)* transaction represents a call to the dedicated function on the Tombolo smart contract and can be sent by any participant. Additionally, its validity is contingent on the *Confirm Funding* transaction not being committed.

For example, if β realizes that although he has confirmed the funding in his blockchain, he cannot proceed further as the funding proposal has reached the time period T in α's blockchain. To discard the formation of the payment channel in β's blockchain, β can send the FD_1 transaction by calling a dedicated function on the Tombolo smart contract in α's blockchain. Committing FD_1 will release the locked funds for α. After FD_1 is committed, β can build its Merkle proof and present it in its own blockchain network as an FDP_2 transaction. This would release the locked funds for β. By implementing these measures, participants are able to ensure that the formation of the payment channel is successful and that any potential loss and/or indefinite locking of funds is avoided. Alternatively, it is also possible for α to commit CF_2 in B_2 in case β goes offline temporarily, as transactions CF_1 and CF_2 can be performed by either participants. This entire channel creation process is illustrated in the state diagram shown in Figure 3.

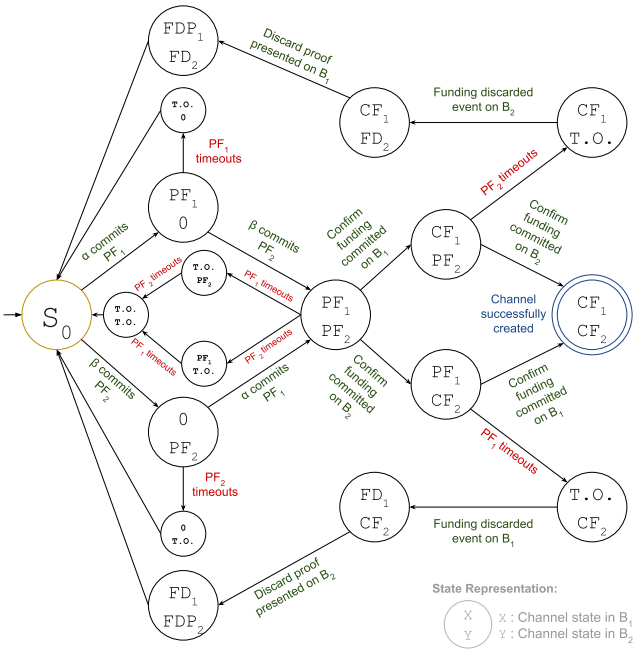

Fig. 3: State diagram for channel creation process in the Tombolo protocol

D. Off-Chain Transactions

The establishment of a successful CCPC enables participants to engage in off-chain transactions by exchanging updated balance proofs. This mechanism bears resemblance to off-chain transactions in existing payment channel systems like Raiden. However, the key distinction lies in the number of balances updated during an off-chain transfer through CCPC. Unlike traditional payment channels, where two balances are updated in the off-chain state, the CCPC may involve updating up to four distinct balances for off-chain transfers.

In this section, we will illustrate off-chain transfers using an example before generalizing the concept. As depicted earlier in Figure 2, the initial state of the channel is $S = \{\alpha : (10, 0),\ \beta : (0, 5)\}$. When α intends to transfer an amount of $3\,BU_1$ (representing a unit of tokens on Blockchain B_1) to β, they will send a signed balance proof with nonce 1 to β, containing the updated state $S' = \{\alpha : (7, 0),\ \beta : (3, 5)\}$.

Moreover, let's consider that β wants to transfer the amount of 6 BU_2 (representing a unit of tokens on Blockchain B_2) to α. If we assume $1BU_1 = 1BU_2$, β can send a signed balance proof with nonce 2 to α, containing the updated state $S'' = \{\alpha : (8, 5), \beta : (2, 0)\}$. It's worth noting that in this transaction, β didn't have a balance of 6 BU_2 on blockchain B_2, so they transfer $5\,BU_2$ to α on blockchain B_2, and the transfer of the remaining $1\,BU_2$ is made on blockchain B_1. This is possible because we assume that both participants hold an address on both blockchains. However, in a real world scenario, there may be a case where $1BU_1 \neq 1BU_2$. To address this, we use the concept of an exchange rate to convert an amount on one blockchain to another. Let's denote R as the exchange rate. Assuming $1\ BU_2 = (1 * R)\ BU_1 = 0.5\ BU_1$ with $R = 0.5$, β would end up sending a total amount of $5\ BU_2 + 0.5\ BU_1$ to α. This demonstrates that users can perform transfers using tokens on both blockchains, totaling the amount of tokens they aim to transfer. This is, however, restricted by the amount of tokens available in the channel. Next, let's generalize this scenario.

Lets consider, α wants to make a payment of $x'\ BU_1$ to β, he can send a signed balance proof to β containing a state as shown in *State 1a* in the Figure 1. Alternatively, α can calculate the equivalent exchange value of $x'\ BU_1$ tokens to some $y'\ BU_2$ using an exchange rate R, then he can send a balance proof containing *State 1b* as shown in the Figure 1. If $1\,BU_2 = (1 * R)\,BU_1$, then $y'\,BU_2 = (x'/R)\,BU_2$. In a more general case, if α wants to make a transfer of $x'\,BU_1$, he can send $x''BU_1$ tokens and $y''BU_2$ tokens, where $x'' + (y'' * R) = x'$.

The parties engaged in the exchange can execute multiple transactions by sharing a balance proof reflecting the updated state without interacting directly with the underly-

ing blockchain networks. Eventually, after completing their transactions, they can close the channel by committing the latest state, distinguished by the larger nonce, which will be discussed next.

E. Channel Termination/Closure

Participants in the channel have the option to exit at any time. However, the process of closing the channel and returning funds to the blockchain must be executed precisely to avoid inconsistent states. This includes scenarios where the channel closes on one blockchain network while remaining open on the other. Another concern is the commitment of the latest state in one blockchain network while closing with the old state on the other blockchain network. Furthermore, either participant may become inactive for some reason and may remain inactive for a longer duration, potentially causing the funds of the active participant to be locked until the other participant becomes active. In this section, we explore two termination cases: the first where both participants are active and the second where one participant is inactive for a longer duration. Additionally, for each case, we shed light on malicious scenarios where a participant attempts to commit an old state and/or experiences channel closure on one blockchain while failing on the other participant's blockchain.

1) Both Participants Active: In a case where both participants agree to close the channel cooperatively, they can initiate a *Propose Withdraw (PW)* transaction in their respective networks, using the balance proof of latest state. Followed by this, it is required to commit the Merkle proof of the PW committed at the other blockchain network involved in the exchange within time T. For example, as depicted in Figure 4, the Merkle proof of PW_1 should be committed before time T. We called the withdrawal is partially completed when the Merkle Proof of PWs are committed on the respective blockchains and the transaction committing them is termed as *Partial Confirmation (PC)* transaction.

However, there may be the chance that one participant has successfully committed the proof of proposal before the time period T on their blockchain while other user failed to do that. To avoid such inconsistencies, a final confirmation using *Confirm Withdrawal (CW)* transaction must be provided on both blockchains, which includes the Merkle proof of the *Partial Confirmation* from the other participant's blockchain network. For example, CW_2 encapsulates the Merkle proof of PC_1, as shown in Figure 4. This two-step confirmation ensures that both participants have satisfied the necessary conditions to withdraw funds from the payment channel without leaving any partially committed withdrawals, thereby maintaining the integrity of the system. Upon the successful attainment of final confirmation on both blockchains, the residual balance resulting from various transactions-representing the funds reflected in the most recent state-is released and becomes available for other purposes, such as conducting additional transactions. It is important to note that transactions PC_1,PC_2,CW_1 and CW_2 can be committed on their respective blockchains by either participant if required.

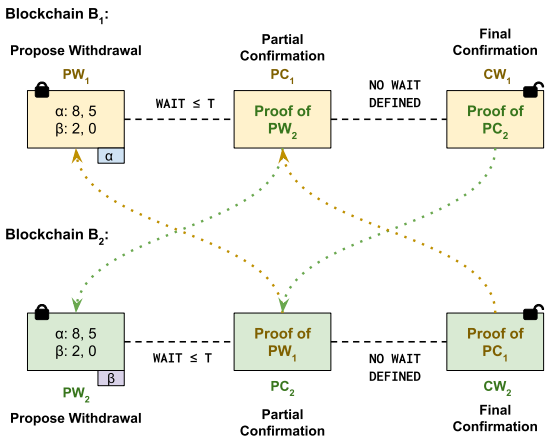

Fig. 4: Channel termination when both participants are active and successfully produce the Propose Withdrawal proof before time T

Figure 4 illustrates a scenario where both participants successfully commit the proof of their respective PWs in their blockchains within time T. The challenge arises when participants fail to commit the corresponding PW proof within the designated time frame T. In a situation where PC_2 transaction containing Merkle proof of PW_1 could not be committed within time T, although the proof of PW_2 was presented in B_1 within time T, the final confirmation cannot be reached as it is required to commit the proof for PC_2, which was never committed in B_2.

In such a scenario, α or any participant can initiate a transaction WD_2 by invoking a dedicated smart contract function. This action generates the *Withdraw Discard* event on β's blockchain (B_2). Subsequently, after WD_2 is committed to B_2, α presents the Merkle proof of WD_2 in the WDP_1 transaction on its blockchain (B_1) and cancels the *Partial Confirmation*. This ensures that both networks remain consistent, even if one participant fails to adhere to the protocol. Due to the absence of waiting time after the *Partial Confirmation*, participants have the flexibility to execute relevant actions at their discretion. This entire collaborative channel termination process is illustrated in the state diagram shown in Figure 5.

2) One Participant Active: When one participant, α in our example, decides to close the channel, when the other participant, β in our example, is observed to be inactive, α will send an *Initiate Withdrawal (IW_1)* transaction on its blockchain (B_1), which is similar to the *Propose Withdrawal*, that includes the latest state's *balance proof*. We denote two different nomenclature to distinguish between two scenarios. Because another participant, β, is inactive, α should initiate the channel closure process at β's blockchain (B_2) by sending a *Request Withdrawal (RW_2)* transaction which includes latest balance proof as used in IW_1 and also the Merkle proof of IW_1, as illustrated in Figure 6.

To confirm a withdrawal on the other blockchain, a *Confirm Withdraw (CW)* transaction must be executed after a minimum of k blocks from the time the *Request Withdrawal*

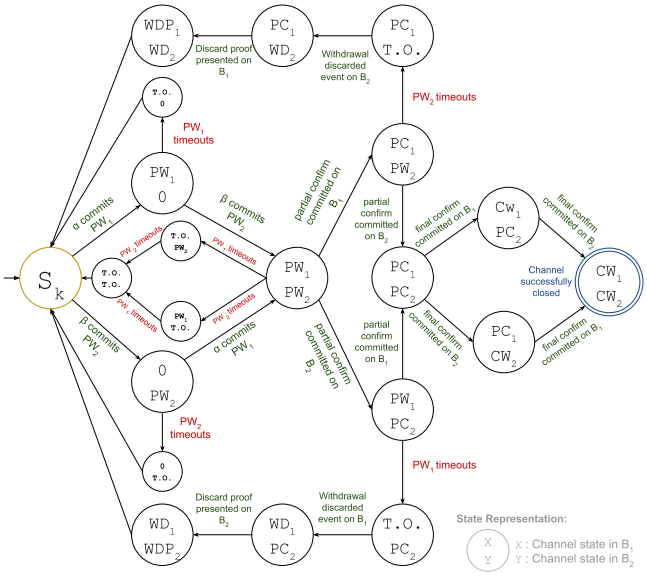

Fig. 5: State diagram for channel termination process when both participants collaborate in the Tombolo protocol

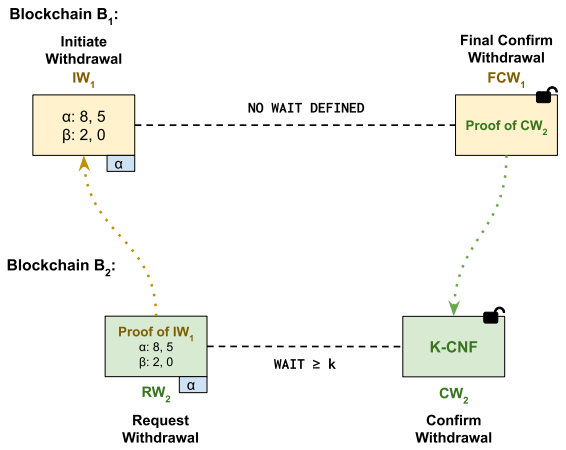

Fig. 6: Channel termination when only one participant is active and close the channel with a latest state's Balance Proof

(RW) transaction was committed. For instance, as illustrated in Figure 6, to validate RW_2 on β's blockchain B_2, α initiates transaction CW_2 on B_2. This transaction needs to be committed on B_2 at least after K blocks from the block where RW_2 was committed. The waiting period ensures that the other participant, β, has the opportunity to propose a more recent *Balance Proof* if one exists. Once the CW_2 transaction is committed on β's blockchain (B_2), the funds from the channel on B_2 are released and deposited at the respective participants' addresses, leading to the closure of the channel.

Now, participant α can release their funds from the channel on their blockchain, B_1, by providing proof that they have waited sufficiently for β to challenge the withdrawal. This is accomplished by submitting a *Final Confirm Withdrawal* (FCW_1) transaction, which is a call to a dedicated function in the smart contract. This function requires the Merkle proof of CW_2. After ensuring the validity of FCW_1, it is committed to blockchain B_1. This action results in the release of funds from the channel on B_1, which are then deposited at the respective participants' addresses, leading to the permanent closure of the channel.

If the active participant α maliciously submits an older *Balance Proof* in IW_1 and RW_2, and the other participant β becomes active within K blocks of RW_2, β can challenge RW_2 with the latest *Balance Proof* by sending a *Challenge Withdrawal (CHW$_2$)*. The commitment of CHW_2 triggers a *Withdrawal Discarded (WD$_2$)* event, preventing the smart contract from accepting any *Confirm Withdraw* transaction (CW_2). Consequently, α cannot achieve final confirmation (i.e., commit FCW_1) on its blockchain.

However, it is crucial to update this channel state in α's blockchain, which can be done with the Merkle proof of the WD_2 event. This proof serves as evidence of rejection, terminating the indefinite wait for final confirmation in α's blockchain B_1. This is accomplished by sending the *Withdrawal Discard Proof (WDP$_1$)* transaction on B_1. This entire process of channel termination with only a single participant online is depicted in the state diagram shown in Figure 7.

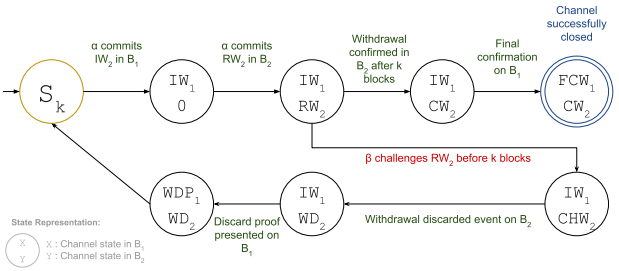

Fig. 7: State diagram for channel termination process when only one participant (α) initiates withdrawal in the Tombolo protocol

IV. Security Analysis

In this section, we present a formal security analysis of the Tombolo protocol, demonstrating that it upholds critical security properties, including atomic state consistency and liveness, even in the presence of offline participants, which can be a deliberate malicious behavior.

Atomic state consistency ensures that during channel creation and termination, both blockchains either reflect the same final channel state or remain unaltered without any committed state. Liveness ensures that the protocol won't lock the user's token forever, even if a node is deliberately or unintentionally deactivated after the channel has been created.

Lemma 1. *The Tombolo protocol ensures atomic state consistency during channel creation in the presence of deliberate or unintentional deactivation of users.*

Proof. Let S_0 denote the initial state of the Tombolo smart contract on B_1 and B_2, which are initially identical. Let S_1

and S_2 represent the channel states after creation on B_1 and B_2, respectively. To prove by contradiction, assume $S_1 \neq S_2$.

For successful channel creation on B_1, PF_2 must be committed on B_2, followed by CF_1 on B_1 within T blocks, updating the state to S_1. Similarly, PF_1 on B_1 and CF_2 on B_2 to update the state to S_2.

For $S_1 \neq S_2$, one of PF_1, CF_2, PF_2, or CF_1 must be missing. We prove that channel creation in Tombolo is impossible without all these transactions, leading to a contradiction.

CF_1 cannot commit without PF_2, and CF_2 cannot commit without PF_1. If either PF fails, the funding transaction times out, unlocking funds and reverting both blockchains to the initial state S_0.

If PF_1 and PF_2 commit but either CF_1 or CF_2 fails within timeout T, a *Funding Discarded* event unlocks funds on one chain. Its proof (*Funding Discard Proof*) unlocks funds on the other chain, reverting both blockchains to S_0, a contradiction

Finally, if all transactions are committed on their respective blockchains within the time limit, PF_1 and PF_2 update the state based on the tokens mutually agreed upon off-chain by α and β, resulting in $S_1 = S_2$, which contradicts the assumption.

It is important to note that a proof of incorrectly committed PF (where contribution values do not match with the corresponding PF on the other chain) will not be accepted by the Tombolo smart contract on the other chain, effectively rendering it equivalent to a missing (uncommitted) PF.

□

Lemma 2. *The Tombolo protocol ensures atomic state consistency during channel termination in the presence of deliberate or unintentional deactivation of users.*

Proof. We prove this by contradiction. Let S_1' and S_2' represent the states on B_1 and B_2 after termination. Assume, for contradiction, that $S_1' \neq S_2'$.

To terminate the channel on B_1, PW_1, PC_2, and CW_1 must be committed in order on B_1, B_2, and B_2, respectively. Similarly, for termination on B_2, PW_2, PC_1, and CW_2 must be committed in order on B_2, B_1, and B_2, respectively.

If both users include the latest state (reflected by the transaction with the highest nonce), Lemma 1 ensures that the state on both blockchains after channel creation, S_0', is identical. Thus, the resulting state after termination will also be identical.

For $S_1' \neq S_2'$, one user must terminate with an older state (reflected by a transaction with a lower nonce), or one user must commit all transactions on their blockchain while the other misses some.

In the first case, where one user attempts to terminate with an older state in PW, the other user will reject it by not including their Merkle proof in PC, aborting the termination and reverting to the post-channel creation state, which is identical on both blockchains (from Lemma 1).

If one user fails to commit transactions, the Final Confirmation can only fail if the other participant does not commit Partial Confirmation before timeout T. In this case, the *Withdraw Discard* event can be triggered, and its proof is presented to the chain in the Partial Confirmation phase, ensuring both blockchains revert to the post-channel creation state, identical on both blockchains (from Lemma 1). Thus, we reach a contradiction in both cases. □

Lemma 3. *Let f represent the transaction fee associated with any Tombolo transaction. For simplicity, we assume that f is the same on both B_1 and B_2. Furthermore, let v_1 and v_2 denote the values committed by α on B_1 and β on B_2, respectively. If $\frac{\max(v_1,v_2)}{k} < f$, Tombolo guarantees that channel termination with latest state, i.e., the one corresponding to the transaction with the largest nonce, will always be dominant strategy.*

Proof. In Tombolo, a malicious user (α or β) may bribe consensus nodes to censor the CHW transaction reflecting the latest state, causing the other user to be treated as inactive for an extended duration. The malicious user achieves this by broadcasting IW on their blockchain and RW on the other user's blockchain, granting the honest user k blocks to challenge the state update.

The malicious user's strategy is to confiscate both v_1 and v_2 by bribing consensus nodes for k blocks. Let v_1 be the malicious user's contribution and v_2 the honest user's contribution. The bribe paid to censor the honest user's CHW is $k * f$. Since $\frac{\max(v_1,v_2)}{k} < f$, there exists at least one block among k blocks where the malicious user is left with a fee less than f.

If $v_1 > v_2$, the malicious user cannot profit from the attack, as it would earn less than v_1. Since $\frac{v_1}{k} < f$, the malicious user spends v_1 and part of v_2 on bribes, earning $v_2 - [f - (v_1 - f(k-1))] = v_2 - (k \cdot f - v_1)$. Given that $k \cdot f - v_1 > 0$, the malicious user earns less than v_1, their own contribution.

If $v_1 < v_2$, the malicious user pays a bribe of $v_2 - f(k-1)$ for $k-1$ blocks and takes share from v_1 to cover the k-th block fee. Ultimately, the malicious user earns less than v_1.

Lastly, any attempt to post an old state using PW will be rejected by the honest user, who will include the Merkel proof in PC and subsequently in CW.

Note that, ideally, the malicious user must pay more than f to censor the honest user's transaction; however, for simplicity, we consider f.

□

Theorem 1. *The Tombolo ensures atomic consistency by committing the latest state, even in cases of intentional or unintentional deactivation of a participating user.*

Proof. From Lemmas 1 and 2, Tombolo ensures consistency. Additionally, Lemmas 2 and 3 demonstrate that while a user may deviate from the protocol to terminate a channel with an old state, they incur a token loss compared to terminating with the latest state. Despite such attempts, Tombolo ensures the same state is committed on both blockchains, maintaining consistency. □

Theorem 2. *The Tombolo protocol ensures liveness for active participants, even if the other user is deactivated or attempts to censor active user's transactions to misclassify them as inactive.*

Proof. We prove this with a specific example, which can be adopted to a generic scenario. Let α be the active participant and β the inactive one. α initiates channel termination by sending *Initiate Withdrawal (IW)* on B_1 and *Request Withdrawal (RW)* on B_2 with the latest state and Merkle proof of IW.

After K blocks, if β does not respond, α finalizes the withdrawal with *Final Confirm Withdrawal (FCW)*. If β challenges RW with a newer state, the withdrawal is discarded. So The active user can always unlock the tokens in the channel, and the respective state is updated on both blockchains.

□

The Tombolo protocol satisfies atomicity, consistency, and liveness by leveraging Merkle proofs, and well-defined timeout mechanisms. These guarantees ensure robust security against adversarial and failure scenarios.

V. Implementation and Experiments

In this section, we present the implementation and evaluation of the Tombolo protocol. We start by describing the experimental setup, including the implemented smart contracts, their use of ETH Relay [15] for Merkle proof verification, and the parameters and metrics for evaluating Tombolo. We conclude with the experimental results.

A. Experimental Setup

The functional implementation of the Tombolo protocol integrated with an implementation of a traditional payment channel protocol for EVM (Ethereum Virtual Machine) based blockchains, has been designated as *Cross Network Payment Channel System (XNPCS)*. We have evaluated XNPCS using two distinct blockchain networks connected through a relay network, as illustrated in Figure 8. Additionally, all experiments were conducted assuming the absence of malicious nodes, ensuring the continuous availability of relayed block headers for immediate utilization. Furthermore, all reported experimental results are based on five independent runs. To gain a deeper insight into the experimental setup, let's delve into the various components of XNPCS.

1) XNPCS Smart Contracts: XNPCS includes two smart contracts for payment channels written in Solidity v0.8.0 [26] : XPC and SPC. The XPC smart contract is an implementation of functions required to build cross-chain payment channels following our Tombolo protocol, while the SPC smart contract includes functions required for a Simple Payment Channel system, which enables users to build intra-chain payment channels. SPC essentially represents Raiden-like traditional payment channel implementations. SPC is primarily developed to conduct experiments and facilitate easy integration of intra-chain payment channel network with our Tombolo protocol based cross-chain payment channels. Furthermore, XNPCS includes a web application client that simplifies interacting with relevant smart contracts with the help of Web3 [27]. The XNPCS client also enables users to connect via peer-to-peer connections using WebRTC [28].

Figure 8 provides an architectural overview of the XNPCS implementation, illustrating the placement of XPC and SPC. In particular, it depicts the establishment of a cross-chain channel between user C on Blockchain B_1 and user D on Blockchain B_2. Subsequently, we will delve into the interaction between XPCs during channel creation and channel termination.

2) Channel Creation Process in XNPCS: The channel creation phase consists of three steps: initialization, self-funding, and verification. During initialization, both participants register a new channel with zero balance in their respective XPC smart contracts. Following this, each participant performs a self-funding transaction to add their contribution to the channel. These two steps together represent the *Propose Funding* phase in the Tombolo protocol (III-C2), where channel states in the two blockchains remain independent.

Next, the block headers containing the self-funding transactions are forwarded to the ETH Relay smart contract on the other blockchain by relay node. Upon receiving these block headers, participants initiate the verification of the self-funding transactions from the other blockchain within their XPC smart contracts. The XPC contract uses the ETH Relay contract to validate the Merkle proof of the transactions. Successful verification confirms the presence of the self-funding transaction in the other blockchain, completing the *Confirm Funding* phase (III-C2).

3) Channel Closure Process in XNPCS: The channel closing phase involves three steps: proposal, partial confirmation, and final confirmation, as discussed in III-E1. Both participants initiate the process by submitting a proposal transaction with the latest balance proof to the XPC smart contract on their respective blockchains. They then wait for the block header containing the proposal from the other blockchain. Upon receiving it, each performs a partial confirmation transaction in their blockchain, including the Merkle proof of the proposal from the other blockchain. Finally, after the block header containing the partial confirmation arrives, each participant submits a final confirmation transaction with the Merkle proof of the partial confirmation. This concludes the channel closing process, transferring funds from the XPC smart contract to the respective accounts.

4) Metrics: We have evaluated XNPCS using the following metrics: 1) Gas Cost, 2) Channel creation time, and 3) Channel termination/closing time. The gas cost for a process, such as channel creation, is calculated by adding up the gas used in all intermediate smart contract functions required to complete that particular process in a single blockchain.

B. Results

We compared SPC and XPC with Raiden, a state-of-the-art protocol, and found that SPC (our implementation of the intra-channel protocol) for channel creation and closure incurs a similar gas cost compared to Raiden. However, as illustrated in Figure 9, XPC incurs almost 2.5 times the gas

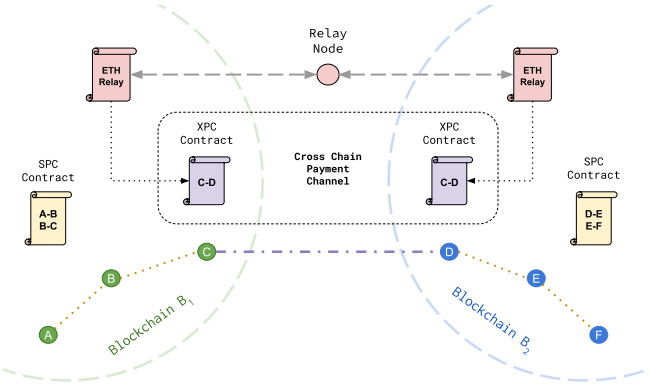

Fig. 8: XNPCS architecture, and involvement of ETH relay node and ETH relay smart contract in connecting two blockchain networks, B_1 and B_2

cost for channel creation and around 3.5 times the gas cost for channel termination compared to Raiden. The additional time required for channel termination is due to the process involving three steps (withdrawal proposal, partial confirmation, and final confirmation), while the channel creation phase involves two steps (initialization+self-funding and verification). In other words, channel termination requires three smart contract calls, while channel creation involves two. Furthermore, the same figure highlights a comparison with CrossChannel, where our protocol reduces channel creation costs by approximately 26%, albeit at a higher cost for channel closure. Nevertheless, it is important to note that the frequency of channel creation is expected to surpass that of channel closure. An intuitive analogy can be drawn from bank accounts, where accounts are opened far more frequently than they are closed.

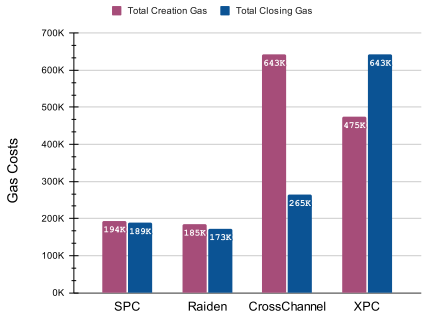

Fig. 9: Gas Cost Comparisons: SPC, Raiden, CrossChannel and XPC

Furthermore, we calculated the time required for each step during the channel creation and channel termination phases. Our observation reveal that the last step, i.e., the verification in channel creation and the final confirmation in channel termination, significantly contributes to the overall time required for the phase to complete, as depicted in Figures 10 and 11, respectively. The significant contribution of the last step is due to the fact that, starting from the third block onward, the relay node requires a block inter-arrival time to deliver a block to another blockchain. Additionally, we noticed a linear growth for each component (i.e., the time required for each step) in both channel creation and channel termination with the block inter-arrival time, as illustrated in Figures 10 and 11. Consequently, the same behavior is observed for the overall time required for channel creation and channel termination, as depicted in Figures 12 and 13. We observed that XPC consistently takes about twice as long as SPC for both channel creation and termination, regardless of the block inter-arrival time, as shown in Figures 12 and 13.

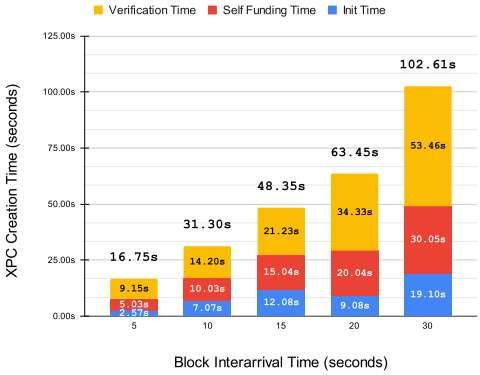

Fig. 10: Variation of time required for various steps involved in the XPC channel Creation with the block inter-arrival time

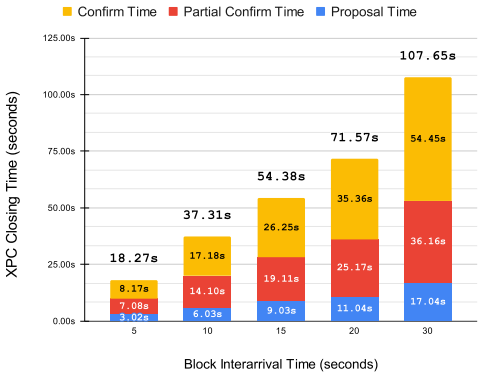

Fig. 11: Variation of time required for various steps involved in the XPC channel termination with the block inter-arrival time

Anticipated factors, including the waiting time for block headers from the other network and additional gas usage for on-chain Merkle proof verification, are expected to result in increased time and gas costs for the creation and termination of cross-chain payment channels. It is crucial to highlight that these costs are one-time expenses, as they will be amortized over numerous off-chain payments once the cross-chain payment channel is successfully established. Our Tombolo protocol enables decentralized cross-chain payment channels but may involve longer setup and termination times and higher gas costs than simple payment channels.

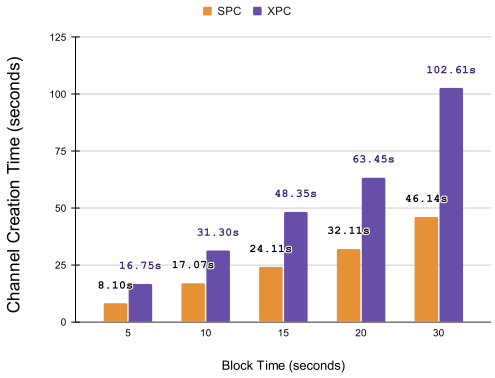

Fig. 12: Variation of SPC and XPC channel creation time with the block inter-arrival time

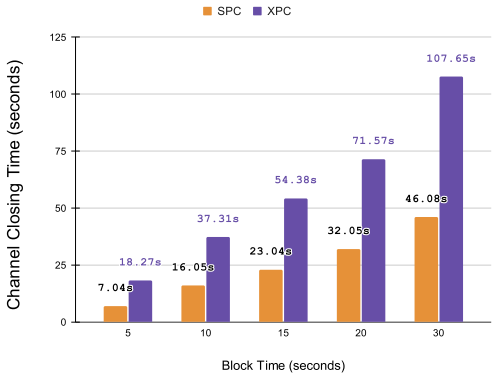

Fig. 13: Variation of SPC and XPC channel termination time with the block inter-arrival time

VI. Conclusion

The Tombolo protocol, introduced in this paper, presents a novel solution for achieving interoperability between disparate blockchain networks . While it incurs slightly longer formation and termination times and higher gas costs in comparison to intra-chain payment channels, these additional expenses become practically insignificant as they get amortized over a considerable number of off-chain payments. As the technology continues to mature, the potential for fostering a more interconnected blockchain ecosystem becomes increasingly evident. Widespread adoption of Tombolo has the potential to unlock new realms of collaboration and inclusivity in the decentralized landscape, laying the groundwork for a more interconnected and resilient future for blockchain technology.

References

[1] K. Narayanam, V. Ramakrishna, D. Vinayagamurthy, and S. Nishad, "Atomic cross-chain exchanges of shared assets," in *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, AFT '22, ACM, Sept. 2022.

[2] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM symposium on principles of distributed computing*, pp. 245–254, 2018.

[3] J. A. Mathus Garza, "The lightning network cross-chain exchange: a decentralized approach for peer to peer exchange across blockchain," *Massachusetts Institute of Technology*, 2018.

[4] X. Jia, Z. Yu, J. Shao, R. Lu, G. Wei, and Z. Liu, "Cross-chain virtual payment channels," *IEEE Transactions on Information Forensics and Security*, 2023.

[5] "Btc relay: A bridge between the bitcoin blockchain ethereum smart contracts." http://btcrelay.org/, 2023. Accessed: 2023-03-13.

[6] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkbridge: Trustless cross-chain bridges made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3003–3017, 2022.

[7] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[8] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[9] "Raiden network." https://raiden.network/101.html, 2022. Accessed: 2022-10-09.

[10] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[11] "Tombolo definition and meaning." https://www.merriam-webster.com/dictionary/tombolo, 2024. Accessed: 2024-01-06.

[12] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*, pp. 369–378, Springer, 1987.

[13] P. Chatzigiannis, F. Baldimtsi, and K. Chalkias, "Sok: Blockchain light clients," in *International Conference on Financial Cryptography and Data Security*, pp. 615–641, Springer, 2022.

[14] X. Luo, K. Xue, Q. Sun, and J. Lu, "Crosschannel: Efficient and scalable cross-chain transactions through cross-and-off-blockchain micropayment channel," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[15] P. Frauenthaler, M. Sigwart, C. Spanring, M. Sober, and S. Schulte, "Eth relay: A cost-efficient relay for ethereum-based blockchains," in *2020 IEEE International Conference on Blockchain (Blockchain)*, pp. 204–213, IEEE, 2020.

[16] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "Xclaim: Trustless, interoperable, cryptocurrency-backed assets," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 193–210, IEEE, 2019.

[17] R. Zarick, B. Pellegrino, and C. Banister, "Layerzero: Trustless omnichain interoperability protocol," *arXiv preprint arXiv:2110.13871*, 2021.

[18] "Wrapped bitcoin (wbtc) delivers the power of an erc20 token! the first erc20 token backed 1:1 with bitcoin." https://wbtc.network/, 2023. Accessed: 2023-03-13.

[19] "Ren: open and community-driven protocol that enables the movement of value between blockchains." https://bridge.renproject.io/about, 2023. Accessed: 2023-07-25.

[20] A. Lazarenko and S. Avdoshin, "Financial risks of the blockchain industry: A survey of cyberattacks," in *Proceedings of the Future Technologies Conference (FTC) 2018: Volume 2*, pp. 368–384, Springer, 2019.

[21] E. Dawson and D. Donovan, "The breadth of shamir's secret-sharing scheme," *Computers & Security*, vol. 13, no. 1, pp. 69–78, 1994.

[22] R. Cramer, I. B. Damgård, *et al.*, *Secure multiparty computation*. Cambridge University Press, 2015.

[23] A. J. Steve Ellis and S. Nazarov, "A decentralized oracle network: Chainlink whitepaper," 2017.

[24] L. Breidenbach, C. Cachin, B. Chan, A. Coventry, S. Ellis, A. Juels, F. Koushanfar, A. Miller, B. Magauran, D. Moroz, *et al.*, "Chainlink 2.0: Next steps in the evolution of decentralized oracle networks," *Chainlink Labs*, vol. 1, pp. 1–136, 2021.

[25] B. Kusmierz and R. Overko, "How centralized is decentralized? comparison of wealth distribution in coins and tokens," in *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–6, IEEE, 2022.

[26] "Solidity programming language." https://soliditylang.org/, 2023. Accessed: 2023-12-26.

[27] "Web3.js: A javascript library for building on ethereum." https://web3js.org/, 2023. Accessed: 2023-07-27.

[28] "Webrtc: Real-time communication for the web." https://webrtc.org/, 2023. Accessed: 2023-06-16.