Bridging assets between the Lightning Network and EVM-compatible blockchains

Tim Käbisch
Hochschule Mittweida, Mittweida, Deutschland

The cryptocurrency ecosystem has seen significant growth with Ethereum and Bitcoin as foundational pillars. Ethereum introduced smart contracts revolutionizing decentralized applications (dApps) across various domains. Scalability challenges led to alternative ecosystems like Binance Smart Chain and Polygon, maintaining compatibility through the Ethereum Virtual Machine (EVM). Bitcoin also faces scalability issues, leading to the Lightning Network's development—an off-chain solution with payment channels for scalable instant transactions. Interoperability is increasingly crucial as the cryptocurrency ecosystem continues to grow, enabling seamless interactions between assets and data across multiple blockchain platforms. EVM-compatible blockchains and the Lightning Network offer unique advantages in their respective use cases. This paper utilizes atomic swaps to create a secure, fast, and user-friendly trustless bridge between the Lightning Network and EVM-compatible blockchains, fostering the growth of both ecosystems and unlocking novel opportunities.

1. Introduction

In the fast-evolving cryptocurrency ecosystem, numerous innovative ideas and blockchain platforms have emerged. Nevertheless, two ecosystems remain the prominent pillars of the cryptocurrency landscape: Ethereum and Bitcoin.

The introduction of smart contracts by the Ethereum network revolutionized the blockchain industry, empowering the development of diverse decentralized applications (dApps) spanning various domains like insurance, decentralized finance (DeFi), social platforms, and games. [1] Yet, scalability challenges persist within the Ethereum network, leading to the emergence of alternative blockchain ecosystems such as Binance Smart Chain, Polygon, and Avalanche. [2] Despite their distinctions, these ecosystems share a common feature - utilizing the Ethereum Virtual Machine (EVM) to modify their networks, fostering compatibility with the broader Ethereum ecosystem. [3]

Bitcoin [4], as the second pillar, stands as a widely-used global system, maintaining a transparent record of transactions on a publicly accessible ledger. However, its scalability encounters challenges due to each participating computer bearing the responsibility of validating, observing, and storing every transaction. Addressing this scalability concern, Joseph Poon and Thaddeus Dryja proposed a solution in their paper *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. [5] The Lightning Network, functioning as a second layer on top of Bitcoin, introduces off-chain payment channels, enabling users to conduct even the smallest transactions, such as micropayments, without the need to publish them on the Bitcoin blockchain. Since its initial proposal in 2015, followed by protocol implementation in 2018, the Lightning Network has garnered considerable attention from developers and investors, with numerous applications currently in development. [6, p. 9-12]

As the cryptocurrency ecosystem undergoes continuous growth and development, the importance of interoperability has become increasingly vital. Interoperability in the context of blockchain technology refers to the seamless interaction of assets and data across multiple blockchains. While exchanging data and value between parties using the same blockchain platform, like Bitcoin, is straightforward, it becomes more complex when different blockchain platforms are involved. EVM-compatible blockchains and the Bitcoin Lightning Network offer distinct advantages in their respective use cases. Establishing interoperability between these two systems by building a bridge holds the potential to unlock new possibilities and use cases, making it a significant focus area for research and development. [7]

Past attempts to construct a bridge between the Lightning Network and EVM-compatible blockchains are evident through various prototypes on GitHub. [8, 9] However, the need to run a separate Lightning node limits its accessibility to the masses. This paper presents the implementation of a secure, fast, and user-friendly trustless bridge between the two systems using atomic swaps to enable a seamless transfer of assets.

2. EVM-compatible blockchains

In the domain of software development, programmers commonly employ high-level programming languages, including Java, Python, or C++. Despite being human-readable, these languages cannot be directly executed by a computer's central processing unit (CPU). Consequently, a crucial process known as compilation is employed to translate the code into machine-executable bytecode. A compiler, a specialized software program, performs this task by converting the high-level code into a lower-level, machine-readable format. Once compiled, the CPU can execute the bytecode, enabling the computer to operate the program as intended.

Blockchain networks, such as Ethereum, operate as decentralized systems across globally distributed nodes. This unique distributed architecture sets them apart from traditional computing systems, as they do not rely on a single central processing unit (CPU) for program execution. Instead, the Ethereum network employs the Ethereum Virtual Machine (EVM) as a software-based CPU to execute bytecode on each network node. This enables developers to write smart contracts in high-level programming languages like Solidity, which are subsequently compiled into bytecode for execution by the EVM. This approach fosters the Ethereum network's operation without the need for a central control point, ensuring a truly decentralized and versatile *world computer*. [3]

When a smart contract is deployed on a blockchain network, it is replicated across all nodes in the network. Users can interact with the contract by submitting transactions, which are executed by each node's Ethereum Virtual Machine (EVM). The EVM ensures that the transactions are processed consistently across all nodes, resulting in a synchronized global state. In essence, the EVM acts as the central component of the distributed state machine, coordinating the execution of transactions to maintain a consistent state among all network participants. [10, p. 297]

3. Lightning Network

The Lightning Network is a revolutionary protocol that facilitates fast and cost-effective online value exchange. Serving as a second-layer technology for Bitcoin, it enables scalable transactions between users, including micropayments for the smallest amounts. Introduced in 2015 and implemented in 2018, the Lightning Network has gained considerable interest from developers and investors, with numerous applications currently in development. [6, p. 1]

3.1 Scaling Bitcoin

Bitcoin, a widely-used global system, faces scalability challenges as each participating computer validates, observes, and stores transactions. The increasing popularity and transaction demand have led to the frequent reaching of the block size limit. When blocks are full, additional transactions queue up, leading to increased fee competition. Consequently, lower-value transactions may become unprofitable during high-demand periods. One solution is to raise the block size limit, allowing more transactions. However, this would shift costs to node operators, requiring greater resources for blockchain validation and storage. Larger block sizes also impose higher bandwidth and processing requirements, leading to increased centralization as fewer individuals can afford to operate a node. [6, p. 9]

Bitcoin's scalability is often compared to Visa, which can process around 40,000 transactions per second at peak usage. [6, p. 9] To match Visa's capacity, Bitcoin would require a block size limit of approximately eight gigabytes, resulting in over one terabyte of transaction data daily with a block mined every 10 minutes on average. This increased size would make running a node at home impractical, limited only to large companies with ample resources. However, even achieving Visa's transaction capacity would only equal traditional financial payment networks and might not be sufficient for future demands with the rise of microtransactions and machine-to-machine payments. [6, p. 9-10]

The paper *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments* by Joseph Poon and Thaddeus Dryja presents a solution to Bitcoin's scalability challenge through the Lightning Network. This second-layer network introduces off-chain payment channels, allowing users to transact without recording each transaction on the Bitcoin blockchain. Payment channels are established as 2-of-2 multi-signature addresses, enabling efficient and unlimited payments within the channel's lifespan. Transactions occur off-chain through signed transactions, and the channel can be closed by broadcasting the latest transaction to the blockchain. [6, p. 10-12, 40]

The Lightning Network's ability to enable instant payments with minimal fees has captured the attention of developers interested in building Lightning Applications (LApps). One promising use case for LApps is pay-per-use, offering users the option to pay solely for the specific services they utilize, rather than subscribing to a fixed plan. This model can be advantageous for various scenarios, such as newspaper subscriptions, where users only pay for the articles they read, or music streaming services, where they only pay for the songs they listen to. Additionally, lightning payments can extend to everyday goods, like purchasing coffee at a restaurant or snacks from a vending machine. Compared to traditional payment methods like cash or credit cards, lightning payments offer faster and cheaper transactions, benefiting both consumers and providers. [11, 12]

3.2 Invoices

Bitcoin transactions involve sending funds to the receiver's Bitcoin address, accessible only with the corresponding private key. This address can be used multiple times without limitations. Conversely, the Lightning Network relies on unique invoices to initiate payments. The recipient generates a unique secret (preimage) for each invoice and must reveal it in the end to complete the payment. To prevent fund theft, a different secret should be used for each payment. Payments on the Lightning Network are atomic, ensuring they are either fully successful or not, with no partial states. Recipients can share invoices through various channels, including email, chat, or QR codes. [6, p. 55, 336]

A lightning invoice contains vital details, such as the payment hash, payment amount, payment description, and expiry time. The payment hash plays a critical role as the

payment identifier and key to finalizing the payment. To generate the payment hash, a unique secret (preimage) is selected and then hashed using the SHA-256 algorithm. The preimage remains exclusively known to the invoice creator, as reversing the SHA256-algorithm is practically impossible. [6, p. 56]

In the Lightning Network, payments are facilitated through hashed timelock contracts (HTLCs), allowing funds to be securely transferred through a route of payment channels from the payer to the recipient. HTLCs ensure that intermediaries cannot steal the funds during the routing process. To finalize the payment, the recipient must reveal the preimage to settle the HTLC along the route. Thus, if the payer possesses the preimage, the payment is completed, and the preimage serves as proof of payment. This concept of utilizing the preimage as proof of payment is crucial to this work and will be further explored in chapter 4. [6, p. 56]

3.3 WebLN

WebLN serves as a set of established guidelines for Lightning applications and client providers, aiming to facilitate secure interactions between web applications and users' wallets. It provides a programmatic interface that empowers applications to make payments, generate invoices for receiving payments, and perform other associated functionalities seamlessly. Initializing and executing WebLN merely necessitates a few lines of JavaScript code, a popular language extensively used for developing web applications. [13]

Alby, a versatile and open-source browser extension, serves as a prominent provider of WebLN. It is purpose-built to provide profound integration between the Bitcoin Lightning Network and web applications. The primary intent of this extension is to ease the web payment process by using the WebLN standard as a bridge between websites and Lightning Network nodes. This interface has significantly simplified the user experience for payments, authentication procedures, and other related functions. [14]

The extension's capabilities make it a perfect fit for the paper’s objective: creating a trustless bridge between the Lightning Network and EVM-compatible blockchains. Utilizing Alby in this context can significantly improve the user experience, enhancing its efficiency and reliability. The implementation of the WebLN standard in Alby enables the sending of payments with just a few lines of code, as shown in figure 1. [14]

```
1   await webln.enable();
2   const invoice = "lnbc10u1pjqw9nnp...";
3   const result = await webln.sendPayment(invoice);
```

Figure 1: Lightning Payment with WebLN

4. Concept

The concept of atomic swaps empowers users to transfer funds between distinct blockchain ecosystems without the need for centralized exchanges as intermediaries. This is accomplished through the use of hashed timelock contracts (HTLCs). However, a prerequisite for this is the compatibility of both ecosystems with the same hashing algorithm. Since both the Lightning Network and EVM-compatible blockchains utilize the SHA-256 hashing algorithm, atomic swaps can be effectively used to bridge assets between these two distinct ecosystems.

Hashed timelock contracts (HTLCs) are a type of smart contract that establish conditional payments between two parties. These contracts combine two fundamental concepts: hashlock and timelock, to ensure the transaction's execution adheres to the agreed-upon terms. A hashlock acts as a constraint that prohibits spending an output until specific data, matching a predetermined hash, becomes available. On the other hand, a timelock restricts a transaction from being executed until a predetermined time or deadline is reached. In essence, HTLCs involve a sender locking cryptocurrency up in a contract and sharing a secret with the recipient following a specific action. The recipient can access the funds if they provide the correct secret within a set time frame (the timelock). If they fail to do so, the funds revert to the sender. [15]

The primary concept of this paper is to construct an HTLC that can be unlocked following payment on the Lightning Network. This approach enables an individual to lock up funds in an HTLC on an EVM-compatible blockchain and generate a lightning invoice for the equivalent value of those funds. Subsequently, the HTLC and invoice are provided to a second individual, who, upon payment of the invoice, gains the ability to unlock the funds from the HTLC. This process effectively enables the exchange of funds on the EVM-compatible blockchain for Bitcoin on the Lightning Network.

The core idea behind this concept is to utilize the payment hash of the lightning invoice as the hashlock for the HTLC. This payment hash is generated by selecting and hashing a unique secret (preimage) for the payment, as described in section 3.2. Additionally, the payer possesses the preimage after completing the payment, effectively making the preimage serve as proof of payment.

Hence, an individual could generate a lightning invoice and employ the payment hash as the hashlock for an HTLC on an EVM-compatible blockchain. The value of the lightning invoice corresponds to the value of the funds locked up within the HTLC. Subsequently, the HTLC and invoice are provided to a second individual. Unlocking the HTLC necessitates presenting the corresponding preimage to the hashlock, which essentially means providing a value that, when hashed using the SHA256 algorithm, matches the hashlock. Since the hashlock used in the HTLC is the payment hash of the lightning invoice, the preimage of the lightning invoice also functions as the preimage of the HTLC. As a result, once the second individual successfully pays the lightning invoice and becomes aware of the preimage, they can unlock the HTLC by presenting this obtained preimage. This concept is illustrated in figure 2.

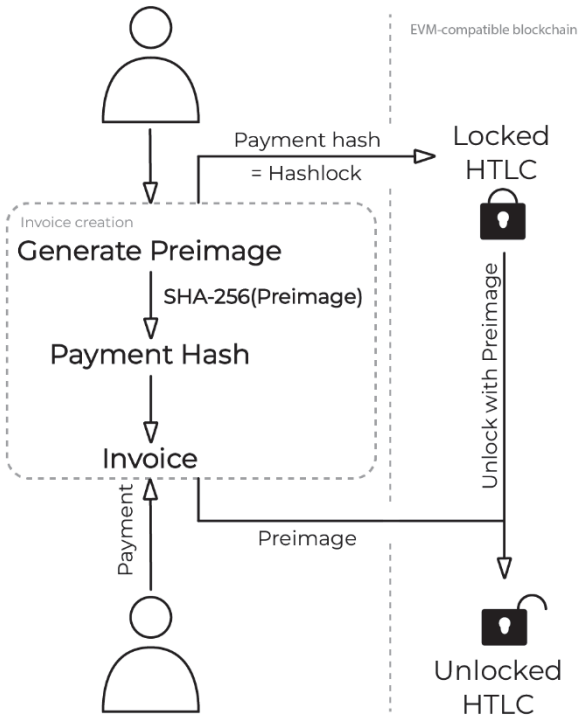

Figure 2: Concept

5. Implementation

In the proposed solution, swaps are always performed between a customer and the operator. However, the design ensures that neither the customer nor the operator needs to place trust in each other. At its core, the implementation consists of three components:

- User Interface (Customer)
- Backend (Operator)
- Smart Contract (HTLC)

For seamless interaction with both an EVM-compatible blockchain and the Lightning Network, a set of tools is required. Users utilize two browser extensions linked to the user interface, as depicted in figure 3. Specifically, Alby is employed for Lightning Network interaction, and Metamask is utilized for the interaction with an EVM-compatible blockchain. On the other hand, the operator utilizes LNbits, a Lightning Network payment management platform, to generate and settle invoices, while leveraging the web3.js library for interactions with EVM-compatible blockchains. [16]

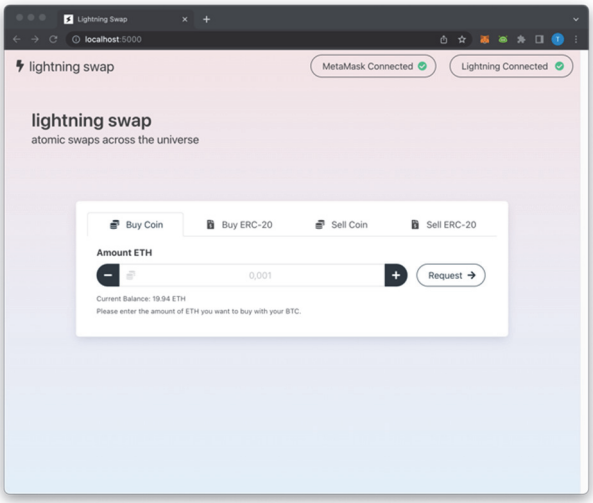

Figure 3: User Interface

The smart contract can be deployed on any EVM-compatible blockchain, enabling a bridge between the Lightning Network and the corresponding blockchain (e.g., Ethereum or Polygon). The term *native coin* is used in this paper to avoid specifying a particular cryptocurrency like ether, as the proposed solution is not limited to the Lightning Network and Ethereum but can be established with any EVM-compatible blockchain. Additionally, it is essential to differentiate between the native coin of a blockchain and tokens. The native coin represents the currency of the blockchain system, such as ether on Ethereum or MATIC on Polygon. In contrast, tokens do not have their blockchain but operate on top of other blockchains. These tokens are usually created in compliance with a specific standard, such as the ERC-20 standard.

This implementation includes four directions: buying native coins and ERC-20 tokens with Bitcoin on the Lightning Network (LN) and selling native coins and ERC-20 tokens for Bitcoin on the LN. While the process of buying native coins with Bitcoin on the Lightning Network is described in detail, the other directions are only briefly discussed in this paper.

5.1 HTLC in Solidity

Implementing a hashed timelock contract (HTLC) within a smart contract is a vital aspect of the proposed concept, which can be accomplished using the Solidity programming language. The HTLC implementation in this paper is based on a library accessible on GitHub. [17] The data for a hashed timelock contract (HTLC) is stored in a *struct*. Among other things, this struct contains elements such as the sender and recipient addresses, the hashlock, and the timelock. Each HTLC's data is stored in

a map, using a 32-byte identifier (ID). To manage HTLCs, the smart contract includes several functions:

a) **haveContract**: This function verifies the existence of an HTLC associated with a given 32-byte ID. It returns a boolean value of true if the HTLC exists and false otherwise.

b) **newContract**: To create a new HTLC, the newContract function is used. It requires three parameters: the intended receiver (who will withdraw funds by providing the preimage), the hashlock, and the timelock. The number of native coins locked in the new HTLC is specified by the msg.value, representing the number of native coins included in the transaction executing the newContract function.

c) **getContract**: Access to HTLC information is possible using the getContract function, which requires providing the HTLC identifier (ID). If an HTLC exists for the given ID, all stored information associated with the HTLC will be returned.

d) **withdraw**: The primary objective of an HTLC is to enable the receiver to unlock and claim their funds by providing the preimage that unlocks the hashlock. The withdraw function facilitates this functionality, taking the HTLC identifier (ID) and the preimage as parameters to claim the funds associated with the HTLC.

e) **refund**: In cases where the intended receiver fails to unlock the HTLC, the creator of the HTLC can reclaim the funds. The refund function allows the creator to call it after the timelock has expired to reclaim their funds.

Although the HTLC mechanism is similar for native coins and ERC-20 tokens, there are slight variations in the implementation for ERC-20 tokens.

5.2 Lightning – Native Coin

This section examines the scenario where a customer acquires native coins using Bitcoin on the Lightning Network (LN). The associated protocol for this situation is visually represented in figure 4. All customers engage in swaps with the operator; however, the design ensures that neither the customer nor the operator is required to place trust in each other.

Before starting a swap, the customer needs to connect their Metamask and Alby wallet to the user interface. They select the network and the number of coins they want to buy with Bitcoin on the Lightning Network, then send an HTTP POST request to the operator. This request, sent to the *offerCoinBuy* endpoint, includes the desired number of coins, the customer's address, and the chosen network.

The operator sends the customer an offer, which contains the details for the swap. Furthermore, it includes a lightning invoice to be paid for accepting the offer (the *offer invoice)*. This protects the operator from spam and unnecessary costs, as the operator bears the gas costs

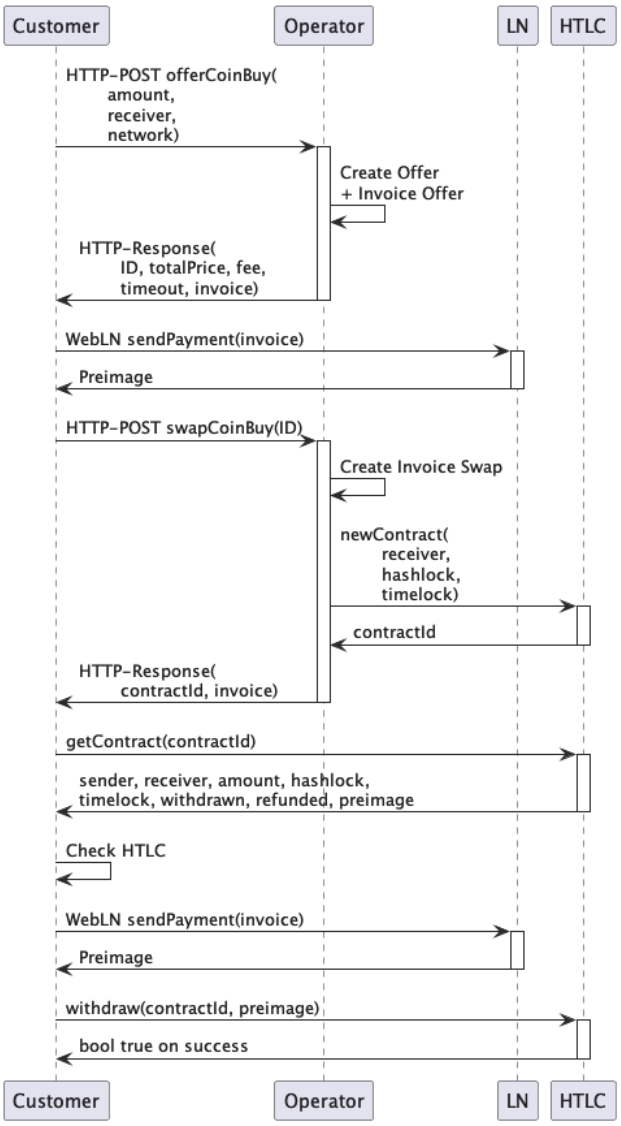

Figure 4: Protocol: Lightning – Native Coin

for creating the HTLC and potential extra costs if the customer fails to conduct the swap. This step prevents customers from requesting a swap without the intention to conduct it. The offer also includes a timeout, which allows the customer time to consider the offer and prevents the operator from committing to a specific exchange rate for an extended period.

The offer is presented to the customer via the user interface. Within the specified timeout, the customer can accept the offer by paying the lightning Invoice using their connected Alby wallet. Upon payment, the user interface sends an HTTP POST request to the operator's *swapCoinBuy* endpoint, including the offer's identifier (ID) to initiate the creation of the HTLC. The operator validates the presence of an offer corresponding to the given ID, verifies that the customer responded within the specified timeout, and ensures the payment of the invoice. If all conditions are met, the operator proceeds to create a lightning invoice for the equivalent value of the requested native coins (the *swap invoice)*.

The payment hash of the lightning invoice serves as the hashlock for creating the HTLC, as explained in detail in

chapter 4. The operator creates a new HTLC by executing the *newContract* function of the smart contract, effectively locking up the native coins. The function returns a 32-byte identifier, the *contractId*, which, along with the lightning invoice, is sent back to the user interface in response to the HTTP request.

Before prompting the customer to pay the lightning invoice, the user interface carries out multiple verification checks. These checks involve verifying whether the invoice's payment hash aligns with the HTLC's hashlock, if the HTLC's timelock is set to at least five minutes in the future, whether the amount locked up in the HTLC matches the requested amount, and if the HTLC's recipient corresponds to the customer's address. The verification check guarantees that the customer does not need to trust the operator blindly. The customer can independently verify all crucial information before paying the lightning invoice of the operator. Upon successful verification, the customer can be confident that after paying the lightning invoice, they can unlock the HTLC and retrieve their funds.

After verifying the HTLC's integrity, the customer pays the lightning invoice using their Alby wallet. Upon successful payment processing on the Lightning Network, the customer knows the payment's preimage (as outlined in section 3.2) and uses it to withdraw their requested native coins from the HTLC. The user interface provides a claim button that initiates a transaction and prompts the customer for confirmation in their Metamask wallet. The transaction calls the smart contract's *withdraw* function, submitting the HTLC's identifier and preimage. The smart contract checks if the preimage aligns with the HTLC's hashlock.

Finally, the smart contract concludes the swap by transferring the coins locked in the HTLC to the customer. Consequently, the customer obtained their desired native coins, and in exchange, the operator received their Bitcoin on the Lightning Network.

Nonetheless, it's important to note that not all swaps may be successfully conducted. If the customer fails to accept the offer, the operator will discard the offer once the timeout period lapses. Should the customer accept the offer but fail to pay the lightning invoice to conduct the swap, the operator must wait for the HTLC to expire due to the timelock. Once it expires, the operator can trigger a refund by invoking the *refund* function of the smart contract. Conversely, if the operator attempts to deceive the customer, the customer will detect this through the verification checks and could simply abort the swap by refraining from paying any invoice.

5.3 Other directions

In addition to purchasing native coins, customers can also buy ERC-20 tokens supported by the operator on the relevant network. While each blockchain has only one native coin, it may support multiple ERC-20 tokens, each identified by a unique address. In contrast to native coins, ERC-20 tokens cannot be directly sent using a regular transaction. Instead, transferring ERC-20 tokens requires utilizing specific functions provided by the ERC-20 token contract.

Hence, the protocol for purchasing ERC-20 tokens, compared to the one for buying native coins, necessitates the following adjustments: the customer is required to specify the address of their preferred ERC-20 token and the management of ERC-20 tokens must be configured accordingly. Besides these modifications, the protocol remains quite similar to buying native coins.

The protocol presented in section 5.2 can also be applied in reverse, enabling customers to sell their native coins and ERC-20 tokens for Bitcoin on the Lightning Network. Similar to the buying process, the customer engages in a swap with the operator, and both parties do not need to trust each other. Upon submitting an HTTP request, the customer receives an offer from the operator, specifying the amount the operator is willing to pay for the native coins or ERC-20 tokens the customer intends to sell.

From this point onward, the customer and operator essentially switch roles compared to the protocol illustrated in figure 4. The customer takes charge of creating the lightning invoice and the hashed timelock contract (HTLC). These tasks are performed programmatically by the user interface, and the customer's role is to confirm the actions in their Alby and Metamask wallets, respectively.

Following this, the customer forwards the lightning invoice and HTLC to the operator. Before paying the lightning invoice, the operator conducts the verification checks outlined in section 5.2. This ensures that the operator does not need to trust the customer. Once the HTLC's integrity is confirmed, the operator pays the lightning invoice, revealing the payment's preimage. Finally, the operator retrieves their native coins or ERC-20 tokens from the smart contract by providing the preimage. On the other hand, the customer received their Bitcoin on the Lightning Network. This protocol enables customers to sell their native coins or ERC-20 tokens for Bitcoin on the Lightning Network in a trustless manner.

6. Conclusion

This paper has shown the development of a trustless bridge that facilitates asset transfer between the Bitcoin Lightning Network and EVM-compatible blockchains. Due to the solution's versatility, it is capable of creating a bridge between the Bitcoin Lightning Network and any blockchain that is compatible with the Ethereum Virtual Machine (EVM). To establish a bridge, the HTLC smart contract simply needs to be deployed on the corresponding EVM-compatible blockchain. The primary focus of this work has been on the technical aspects of the solution. Nevertheless, for the operational deployment of the developed platform, certain non-technical factors, such as economic and security considerations, require further consideration.

Adequate liquidity on the Bitcoin Lightning Network and EVM-compatible blockchain is crucial for the platform's operation. A rebalancing mechanism is needed to prevent trade disruption due to asset shortages. Furthermore, developing a sustainable monetization model, such as charging a competitive swap fee, is essential. Additionally, a comprehensive analysis of costs for customers and the operator, considering the Lightning Network payment route and the demand on the corresponding EVM-compatible blockchain, is required.

Finally, one downside of atomic swaps is the *free option problem*. It refers to a situation where one party can exploit the time delay between the initiation and execution of the swap to gain an advantage. During this time, market conditions may change, allowing the exploiting party to decide whether to proceed with the swap or back out, potentially resulting in an unfair advantage. By implementing a timeout, the free option problem can be mitigated, as is done in the proposed solution. However, determining the optimal duration of the timeout still requires further clarification. [18]

This paper paves the way for the development of a production-ready service that enables a trustless bridge between the Bitcoin Lightning Network and EVM-compatible blockchains using atomic swaps. This research aims to foster the growth of both ecosystems, unlocking new opportunities for them to leverage each other's advantages.

References

[1] Ethereum.org. “What is Ethereum?” (2023), [Online]. Available: https://ethereum.org/en/what-is-ethereum. (last visited on 09/05/2023).

[2] Kearney, Leal. “What are EVM Compatible Blockchains? A Guide to the Ethereum Virtual Machine”. (2023), [Online]. Available: https://blog.thirdweb.com/evm-compatible-blockchains-and-ethere um-virtual-machine/. (last visited on 09/05/2023).

[3] GoCrypto. “What are EVM-compatible blockchains?” (2022), [Online]. Available: https://medium.com/eligma-blog/what-are-evm-compatible-blockchains-64f91c97038e. (last visited on 23/02/2023)

[4] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. (2008), [Online]. Available: https://bitcoin.org/bitcoin.pdf. (last visited on 09/05/2023).

[5] Joseph Poon, Thaddeus Dryja. “The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments”. (2016), [Online]. Available: https://lightning.network/lightning-network-paper.pdf. (last visited on 09/05/2023).

[6] Andreas Antonopoulos, Olaoluwa Osuntokun, René Pickhardt, Mastering the Lightning Network. O’Reilly Media, Inc., 2021, ISBN: 978-1-492-05486-3.

[7] Cointelegraph. “What is blockchain interoperability: A beginner’s guide to cross-chain technology”. (2023), [Online]. Available: https://cointelegraph.com/learn/whatis- blockchain - interoperability - a - beginners - guide - to - cross - chain -technology. (last visited on 19/04/2023).

[8] Leon Do. “submarine-swaps”. (2020), [Online]. Available: https://github.com/leondo/submarine-swaps. (last visited on 26/04/2023).

[9] Aleksey Bykhun. “ln-eth-swap”. (2018), [Online]. Available: https://github.com/caffeinum/ln-eth-swap. (last visited on 26/04/2023).

[10] Andreas Antonopoulos, Dr. Gavin Wood, Mastering Ethereum. O’Reilly Media, Inc., 2018, ISBN: 978-1-491-97194-9.

[11] Blocktrainer. “Bitcoin zum Anfassen: Der Lightning Snackautomat”. (2023), [Online]. Available: https://www.blocktrainer.de/bitcoin-zum-anfassen-der-lig htning-snackautomat. (last visited on 09/03/2023).

[12] Coincharge. “Payment per newspaper article with Bitcoin Lightning”. (2023), [Online]. Available: https://coincharge.io/en/payment-per-newspaper-article-withbitcoin-lightning. (last visited on 09/03/2023).

[13] Alby. “WebLN Guide”. (2023), [Online]. Available: https://github.com/getAlby/webln-guide. (last visited on 06/03/2023).

[14] Alby. “lightning-browser-extension”. (2023), [Online]. Available: https://github.com/getAlby/ lightning-browser-extension#lightning-web-extension. (last visited on 06/03/2023).

[15] Minima. “Understanding Hashed Time-locked Contracts (HTLCs)”. (2022), [Online]. Available: https://www.minima.global/post/understanding-hashed-time-lockedcontracts-htlcs. (last visited on 17/03/2023).

[16] LNbits. “Free Open-Source Bitcoin Lightning Accounts System with Extensions”. (2023), [Online]. Available: https://lnbits.com. (last visited on 09/03/2023).

[17] C. Hatch. “hashed-timelock-contract-ethereum”. (2021), [Online]. Available: https://github.com/ chatch/hashed-timelock-contract-ethereum. (last visited on 26/04/2023).

[18] M. Hammond. “Blockchain Interoperability Series: Atomic Swaps”. (2019), [Online]. Available: https://medium.com/@mchammond/atomic-swaps-eebd0fa8110d. (last visited on 28/07/2023).