Cryptocurrency: User-Friendly Micropayment Channels

Medha Upadhyay mupadhya@ucsd.edu
Isa Vidanes ividanes@ucsd.edu
Minh Luc mluc@ucsd.edu
Priscilla Hui pyhui@ucsd.edu

Sheffield Nolan
Sheffield.Nolan@franklintempleton.com

Abstract

Although traditional blockchain transactions offer many advantages, the overall process often comes with many burdens when used in the context of small, frequent payments. Repeated payments can become costly and inefficient due to the associated transaction fees as well as the slow processing times. Micropayment channels have become a feasible and efficient way to remedy these drawbacks. Through a micropayment channel, individuals can carry out multiple transactions without having to make multiple commits on a blockchain, minimizing the cost for the entire process while making it faster. With the current limitations of existing micropayment channels, we propose to create a user-friendly system for individuals to send small repeated payments to trusted parties.

Code: https://github.com/medhaupadhyay/Micropayment-Channel

1 Introduction

The increasing popularity of blockchain technology has led to the emergence of micropayment channels and they have since become a useful way to manage frequent or repeated transactions in a more efficient and trustless manner. The ability to carry out payments off-chain (via micropayment channels) allows users to engage in more secure, decentralized transactions (Solidity 2018). In spite of these advantages, micropayment channels can have some notable drawbacks, especially in current implementations. One issue is the accessibility of certain models. Some models are not beginner friendly and require prior knowledge as well as usage of blockchain technology and cryptocurrency. We propose to create a more user-friendly system that will enable users to conveniently verify and publish code and carry out transactions with full transparency.

Existing payment channel models come with limitations that can be remediated through our proposed methods (Bogan 2018). In order to create a more accessible model, we plan to design a simple and easy-to-follow interface. This interface will include features, such as opening a payment channel, sending off-chain transactions from sender to recipient, and closing a payment channel. All of these proposed functionalities will be integrated into one web page where users can interact with the features with ease.

Micropayment channels are useful in many different situations and contexts. They, most notably, come in handy when users want to carry out multiple transactions without making multiple commits to a blockchain. The ability to execute payments off-chain and complete the entire transaction process via the blockchain when all is said and done is a convenient as well as effective way to do so, since every transaction on the blockchain comes with a considerable fee. Some possible uses of micropayment channels are making repeated purchases from a store or sending payments to social media influencers.

Transactions on micropayment channels will have the added security that blockchain technology can offer, such as cryptography and node validation (VOYIX 2022).

2 Methods

2.1 Prerequisites

Make sure you have a functioning MetaMask account set up. The account should have Ethereum (we will be using testnet Ethereum for the purposes of this project) otherwise you will be unable to deploy or use the smart contract.

2.2 Deploying the Smart Contract Traditionally

These are the full instructions to deploy the type of smart contract that we will be using. Our micropayment channel provides features to streamline this process and make it easier

for casual users.

Code Setup

- Go to this link to access the template code (Solidity 2018)
- On the right, click “Open in Remix”
- On the top left, make sure the compiler is on the latest version (0.8.23 in our case)
- Click the blue “Compile Contract” button on the left
- On the left menu, click on the Ethereum logo (hovering over the icon reveals “Deploy and Run Transactions”)
- Set environment to “MetaMask” and log in to your MetaMask account when prompted
- Make sure the account address matches the MetaMask account address that you want to use; your account will be charged to deploy this smart contract
- Click the red ”Deploy” button
- Confirm the transaction on the MetaMask pop up

Verifying and Publishing the Code

- Go to MetaMask and click on the transaction that just happened
- Next to status, click on “View on Block Explorer”
- Click on the address listed in the “To” field
- Next to the blue “Transactions” button, there is an “Contract” button; click on it
- Click “Verify and Publish” to start publishing the code
- Set compiler type to “Solidity (Single File)”
- Set the compiler type to match the Solidity compiler you used (0.8.23 in our case)
- Set the license to “No License”
- Copy and paste the code when prompted to fill in the code
- Click “Verify and Publish”

To Use the Contract

- On Etherscan, click on the ”Contract” button (next to the blue ”Transactions” button)
- Click on the ”Write Contract” button (next to the blue ”Code” button)
- Click ”Connect to Web 3” and choose MetaMask when prompted
- Increment the methods as you wish
- Click ”Write” and the transaction should go through.

2.3 Using Our Micropayment Channel

Creating a New Channel

- Launch index.html and styles.css locally on your computer
- Login with MetaMask and connect your account
- Under ”Create New Channel”, enter your MetaMask address as the ”Sender Wallet Address”
- Enter the ”Receiver Wallet Address”; this is the account you will be sending Ethereum to
- Click ”Deploy Contract”

- Check your MetaMask account and click on the transaction that just occurred
- Click ”View on Etherscan” and copy the contract address; keep this for your records

Logging a New Channel

- Enter the contract address of the newly created contract under ”Log a New Channel”
- Click ”Submit”

Logging Payments on an Existing Channel

- Under ”Log Payments on an Existing Channel”, enter the contract address
- Input the amount of Ethereum you would like to send to the receiver
- Click ”Log Payment”

Closing a Channel

- Note: this action is irreversible
- Enter the contract address under ”Close Channel”
- Click ”Close Channel”
- The receiver should receive the Ethereum once the transaction goes through

2.4 Setting Up the Interface

The front-end interface integrates multiple languages – Solidity, JavaScript, CSS, and HTML. This interface allows users to open a new payment channel, send off-chain transactions, and close the payment channel all in one place.

- Convert Solidity into bytecode using the online Remix Ethereum IDE
- Integrate bytecode into JavaScript program (this is where all the pieces connect together – think a machine that runs all the necessary functions and features the user will interact with)
- Use CSS / HTML to style and build the web page

2.5 Deploying and Using the Interface

- Run necessary local files to launch interface
- User must make sure they are logged into MetaMask to retrieve their address and/or the recipient’s address
- Click on ”Open a New Payment Channel” to begin the transaction process
- Enter the addresses into their respective fields
- Enter amount for initial transaction
- Click on ”Send Another Transaction”, if sender wants to send more payments to the receiver (these transactions will be logged off-chain)
- Click on ”Close Payment Channel” once all desired transactions have been logged from sender to receiver

Assuming that both parties are satisfied and there are no more transactions to be made, all the payments logged off-chain will be summed up and sent to the receiver in whole.

2.6 Implementing Our Features

Connecting to MetaMask
Users will be provided a field where they can input their address as well as the receiver's address to connect to MetaMask and begin the transaction process. This was implemented through the integration of Solidity smart contracts (converted to bytecode) and JavaScript, which were used to build the functionalities of the front-end interface. The smart contract acts as the MetaMask connection, while the program written in JavaScript executes the entire process from start to finish.

Opening a New Payment Channel Through Interface
Like the Connecting to MetaMask feature, opening a new payment channel involves the integration of Solidity and JavaScript. The smart contract will initiate the transaction history between the sender and receiver, and open a payment channel between the two parties. (Note: The user will not have to worry about the "contract address", as it will be stored and implemented in the back-end.) Once the addresses are verified as existing addresses, the sender will be permitted to enter an amount that they wish to send to the recipient. A unique signature will be generated and the sender will be expected to store and save this signature for future reference or when the parties wish to close the current channel. Once the payment channel is open, the sender can send more transactions – which will be logged off-chain – as long as needed or until the channel has been closed.

Logging Payments Off-Chain
This feature was implemented using key-value pairs in local storage. Each contract address serves as a key and the amount that the sender owes the receiver is the value. When a new channel is logged, the contract address is registered in local storage with a value of zero. When a new payment is logged, the existing value is increased by the given amount. The new total amount owed is set as the value for that contract key. This value is then displayed to the user in a pop up message, so they are aware of how much they owe the receiver and that the payment was logged.
Along with every logged transaction, a unique signature is generated using web3.js. This signature must be inputted when the channel is closed in order to validate the transaction. Since the total amount due is updated as the information comes in, closing the channel simply involves transferring the recorded dues from the sender to the receiver.

Closing a Payment Channel Through Interface
Closing the payment channel integrates the smart contract and includes a function that allows the user to close the current payment channel. The user must input the unique signature that was generated when the latest payment was logged. It is crucial that the user uses the most recently generated signature in order to ensure that the receiver receives the full payment. After following the prompts, the user will be able to close the channel. The user will also be shown a warning that this action cannot be reversed. Once the payment channel closes, all transactions made from start to end will tally and be sent to the receiver in whole.

3 Results

3.1 MetaMask Integration

The integration with MetaMask ensures that transactions, including contract deployment and interactions, are authenticated and confirmed by the user.

3.2 Successful Deployment

The smart contract can be deployed traditionally or through our interface. Once deployed, the contract will be recorded on the Ethereum blockchain using the MetaMask account specified during the process.

3.3 Logging Payments Off-Chain

Once the contract is deployed, users can log payments off-chain using our interface. The payment logs will be stored locally on their computer, and the running total will be displayed to the user.

3.4 Closing the Channel

Once the sender and receiver decide that it is time to close the channel, they can do so through our interface. This involves sending the amount owed from the sender to the receiver and closing the contract. By only making one lump sum transaction, there will be significantly less fees than having to record each transaction on the blockchain individually.

3.5 Transaction Confirmation

Users confirm transactions through MetaMask, providing an additional layer of security and control over their Ethereum wallet.

4 Discussion

When interpreting the results of our project, we must take into consideration the broader knowledge of smart contract technologies that already exist. First, our smart contract approach aligns with the concepts that were emphasized through previous works - transparency, immutability, user friendly-ness, and security. What sets our approach apart is the more streamlined and accessible process to promote user friendly-ness. Generally speaking,

our approach is similar to the existing technologies, but offers a slight deviation in that it offers users more clarity, which enhances the accessibility of our contract.

Our micropayment channel is hosted on a testnet, which means that it cannot be used to transfer real funds. Another possible limitation is that the contract may not be directly suitable for deployment in the real world. There may be the need to make a few adjustments to the contract before applying it to the real Ethereum blockchain.

Some potential areas of future development are adapting the contract so it can be suitable with other blockchain networks. Currently, the smart contract is compatible with Ethereum, and adapting the contract to work with other blockchain platforms will make it more powerful and accessible to a wider audience.

5 Conclusion

The provided method guides users through deploying a smart contract on the Ethereum testnet using Remix and MetaMask; compiler version and license details are specified for compatibility and code licensing. It emphasizes transparency by encouraging users to verify and publish the contract code on Etherscan. MetaMask integration ensures secure transaction confirmation. Users can interact with the micropayment channel on the Ethereum testnet and use it to sent testnet Ethereum to trusted users.

In essence, this method is a user-friendly guide for experimenting with micropayment channels on the Ethereum test network. With further testing, this framework can be expanded to be deployed on the Ethereum blockchain and be used for real transactions among trusted parties.

6 Appendix

Our project proposal is centered around creating a user-friendly micropayment channel in order to make cryptocurrency transactions easier for casual users. The proposal also details the process of opening and closing a channel, as well as the benefits and limitations of micropayment channels. Once our proposed micropayment channel is created and funded, users can host multiple off-chain transactions without committing to the blockchain every time. Micropayment channels are useful as they preserve transparency and immutability, which not only ensures the transactions are private to the users, but also allows for efficient and low-cost transferring of cryptocurrency.

There are numerous benefits to micropayment channels. Firstly, vendors check the nodes in which the transactions are carried through - whether or not there are sufficient funds or malicious users. This eliminates the need for banks or other singular entities. Furthermore, micropayment channels eliminate the need for the sender to interact with the Ethereum network, and simply make one lump sum on-chain transaction when they decide to close the micropayment; this allows them to bypass the regular transaction fees for each small

transaction and only pay the fee once at the very end.

It is important to discuss the limitations of existing micropayment channels when creating our own. For example, data integrity becomes a concern due to the intermediate transactions being carried out off-chain. This means these specific transactions are not stored on-chain, making them susceptible to unwanted changes. Also, while the channel is open, money from either or both parties must be kept locked in a smart contract, reducing the liquidity of the parties.

References

Bogan, Michael. 2018. "Understanding Counterfactual and the Evolution of Payment Channels and State Channels." [Link]

Solidity. 2018. "Solidity By Example." [Link]

VOYIX, NCR. 2022. "The Creation of Payment Channels and How They Work." [Link]