Exploration of Decentralized Communication

Julien Calonne

Abstract
This project explores the different communication possibilities offered by blockchain technology and peer-to-peer networks, focusing on its advantages in terms of security, confidentiality and ownership. It compares the principles inherent in decentralized networks with current digital communication systems that claim to be entirely confidential. This article discusses the advantages and limitations of two protocols actively maintained and used for decentralized exchange: XTMP (off-chain) and Lens Protocol (on-chain). Finally, the article presents a simplified example of the development of a decentralized on-chain messaging solution on the Ethereum network, including the creation and deployment of a Smart Contract and a dApp. The aim is to gain a better understanding of the technical possibilities currently available and to identify their limitations.

Keywords
Blockchain — P2P Network — Communication — Smart Contracts — dApp

Department of Computer Science and Engineering - University at Buffalo

Introduction

The transmission of information is a very old science, whose first works appeared between the 18th and 19th centuries, when scientists invented the telegraph to transmit coded messages through long distances. Blockchain reinvents technological possibilities by thinking in a different way: we don't trust, we verify. Having for a long time been interested in the ways of communication throughout history, the arrival of blockchain is exciting because it is a new possibility that could redefine our means of communication in the years to come. Everything is to be invented again.

1. Fundamentals of Decentralized Communication

1.1 Core Principles of Blockchain and Smart Contracts

According to Stanford's online class, blockchain is blocks of data linked into an **uneditable**, digital chain. This information is stored in an open-source decentralized environment, in which each block's information is confirmable by every participating computer. It is designed to have **decentralized management** instead of the traditional hierarchical systems we're familiar with. A blockchain is also secured by cryptography which ensure the confidentiality and the integrity of every information.

These characteristics are well reflected in the expectations of digital communication, and blockchain adds the dimension of traceability and therefore non-repudiation : if a user sends a message, he cannot says that he did not do it.

Some blockchains, like Ethereum blockchain, have the possibility to deploy programs on it called smart contracts. Smart contract are immutable programs auto-executed on specific conditions and identifiable by a unique address [1]. Developers can used those contracts to interact with blockchain and create decentralized application as we will see in Part 3.

1.2 Comparison with Traditional Systems

Everything we know about communication today is centralized: whether by mail, e-mail or instant communication. In other words, there is one entity that acts as a link between the various users of a system. This may be the post office in the case of postal exchanges, or large companies in digital ones. In the case of digital exchanges, companies providing applications such as WhatsApp and Telegram use centralized servers to store messages, and their users then face a number of risks, such as the use of personal data for commercial purposes. There is also a small risk of hacking or data theft.

Dependence on a central authority can lead to censorship or outages, as we saw on the March 5, 2023 global outage of the Meta group's social medias. On the other hand, even applications that claim to be totally anonymous, such as Telegram, have recently found their limit: the anonymity of users ends when their managers are taken into custody. In August 2024, the head of Telegram, Pavel Durov, was arrested in France on numerous criminal charges for refusing to reveal the identity of users of his social media. A month later, he announced that he had changed his confidentiality policy to reveal the personal information of users requested by justice. Everyone is free to think that this is a good thing or not, but what is certain is that this would not have been possible with an application whose users data is stored and encrypted on the blockchain without any authority being able to access or modify this information.

1.3 Specific advantages of using peer-to-peer

Decentralized communication could overcome some of the limitations of conventional digital centralized communication.

On the security part, cryptography guarantees message integrity, because data on the blockchain is protected against alteration and deletion, especially as messages are distributed over a global network. In terms of confidentiality, no intermediary has access to messages, and users are identified only by their decentralized identity, a public wallet address, already available and accessible to everyone. It also solves the confidentiality problem mentioned above, as no one can block messages, cut off access or reveal a user's identity. However, as the blockchain is transparent, all a user's actions are visible on it, and could create vulnerabilities enabling their identity to be compromised.

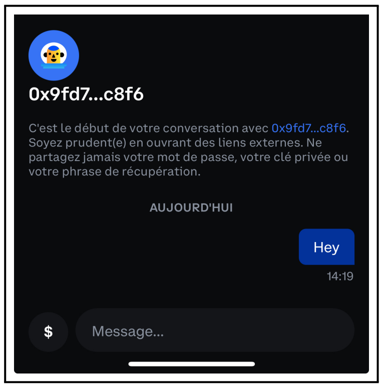
Beyond the security aspects, which can be comparable to highly secure traditional applications, the major advantage of a decentralized application is that, just by knowing his 12 words, a user can use any object connected to Internet to communicate with anyone in the world. He will not need to have a telephone number for which he pays a subscription, create an account on an application or activate multi-factor authentication. We can even imagine that if the decentralized application is no longer available, anyone can create one by knowing the address of the smart contract. We can even communicate directly with the smart contract from IDEs like Remix. These modes of communication therefore offer excellent interoperability.

Communications systems do, however, have their limits, such as the gas costs involved in exchanging messages, or scalability, as we will see in the Limits section of this article.

2. Existing solutions and state of the art

2.1 XMTP Protocol

When asked about existing solutions for online decentralized conversation, one name comes up very quickly: the XMTP protocol. XMTP stands for Extensible Message Transport Protocol, an open-source protocol for web3 messaging whose slogan is three simple phrases: "Pick any app. Message any identity. Own your communications." [2]. In other words, we can choose any application that uses the protocol to retrieve our conversations with any address. Let's see if that's the case.



0x9fd7...c8f6

C'est le début de votre conversation avec 0x9fd7...c8f6. Soyez prudent(e) en ouvrant des liens externes. Ne partagez jamais votre mot de passe, votre clé privée ou votre phrase de récupération.

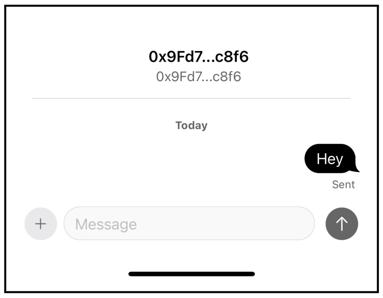
AUJOURD'HUI

Hey

14:19

Figure 1. Conversation example on Coinbase Wallet

We can choose a first application which has a messaging system that works with the XMTP protocol and send a first message. On Figure 1 it is the Coinbase Wallet Application.



0x9Fd7...c8f6
0x9Fd7...c8f6

Today

Hey

Sent

Figure 2. Conversation example on Converse Messenger

We can then choose a second application running on the XMTP protocol (Converse Messenger on Figure 2). Logging in with the same Wallet, we can see that the message sent earlier on the Coinbase Wallet application is indeed present. So the messages are linked to unique addresses and not to a specific application.

If we go a little deeper into how the protocol works, when a user initializes XMTP, a private/public key payload is generated and stored locally. These keys are then associated with an address and a cryptographic signature. Using asymmetric cryptography, when a user sends a message it is encrypted with the recipient's public key, which only the recipient can decrypt. The private key generated and used to decrypt messages is derived from the private key associated with the wallet address, so it can be retrieved by any app.

We might also ask how it is possible that we don't pay any gas charges when we converse, because the XMTP protocol's peer-to-peer network is off-chain [3]. It is a network of nodes, which can be independent or companies that operate the network. The advantage is that messages are not verified, as they are signed by the sender. There is not any charge for sending messages. However, nodes store messages only temporarily, and may disappear after a few days or weeks.

Today, XMTP is used by over 300 applications ranging from instant conversation applications to social medias. Since it is open-source, anyone can create an app using it, free of charge. Other protocols similar to XMTP with their own specifies are also continously developed by the community (Whisper for example [4]).

2.2 Lens Protocol

As we saw earlier, XMTP is a very popular protocol, because it is accessible and does not charge for its use. The downside is that storage is temporary and managed by the nodes, not on the blockchain. Other protocols use the blockchain but charge users for certain functions, as in the case of Lens Protocol [5].

This protocol uses the Polygon sidechain, which is interoperable with the Ethereum blockchain but is designed to solve slowness and high fees problems. It is also compatible with the Ethereum Virtual Machine (EVM) so smart contract can be built and deployed on it. The aim of Lens Protocol is to enable users to own their data in NFT form (their profile, for example). This protocol can be used to create decentralized applications (particularly social medias) where users actions are recorded in the blockchain. Users can then pay for services such as profile creation or gas fees, although these are cheaper on Polygon than on Ethereum.

It is possible for anyone to use Lens Protocol to create dApps in the same way as XMTP, except that it is on-chain.

3. Creation of a Messaging dApp on-chain

In this project, we will focus on on-chain messaging and we are going to use the Ethereum network, in particular the Sepolia testnet on which the smart contract in the example will be deployed. The aim is to demonstrate as simple as possible how to create a messaging dApp in order to better understand communication on the blockchain and its possibilities.

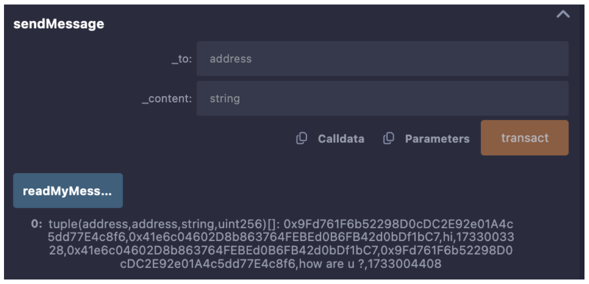
3.1 Smart Contract Design

To create a messaging dApp we need to create a smart contract containing methods in order to write and read information on the blockchain [6]. The smart contract is the backbone of the project, for programmers, it could be assimilated to the back-end part. The smart contract used in this project is called MessageSender and has been deployed on Sepolia network, here is its address : ”0xE88320031C79eaeC940d0768f428C2ab6D61842F”. The language used to developed the smart contract is the most common one Solidity and the IDE used to develop, compile and deploy is Remix.

The smart contract is built around a structure called “Message”, including the recipient’s address, the sender’s address, the content of the message and the time it was sent. Message data is organised by associating a message array with each Ethereum address.

The smart contract has two methods. The first, “sendMessage”, sends a message to the recipient, checking that the message is not empty and that the recipient exists. The message is then stored by the sender and recipient, and an event object is issued on the blockchain to notify the transaction. This method will require a fee to be used because it is generating a new transaction. The second method, “readMyMessages”, enables the user to obtain his received or sent messages. In this way, each user can send and read his or her messages, with access to the history, but without the possibility of modifying or deleting it.

The Figure 3 above displays the possible interaction with the smart contract deployed. To limit the cost of smart contract deployment, it is important to optimize the code by reducing the number of operations and stored variables as much as possible. In the case of the “MessageSender” Smart Contract, the code is reduced to the minimum necessary to make it work. By deploying at a time when gas prices were relatively low, the deployment cost was 0.005 Sepolia. After the first tests, sending a short message like ”hello, how are you?” require to pay 0.0005 Sepolia, which is also relatively low.

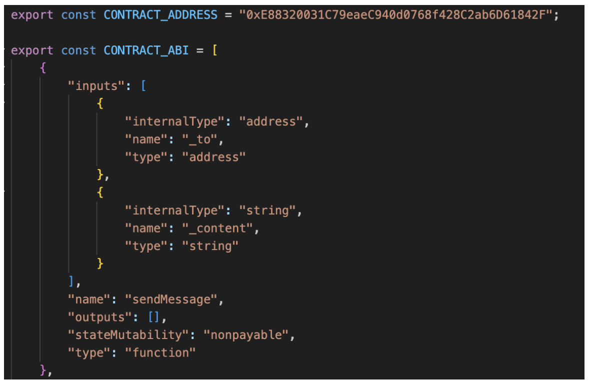


```
0: tuple(address,address,string,uint256)[]: 0x9Fd761F6b52298D0cDC2E92e01A4c
   5dd77E4c8f6,0x41e6c04602D8b863764FEBEd0B6FB42d0bDf1bC7,hi,17330033
   28,0x41e6c04602D8b863764FEBEd0B6FB42d0bDf1bC7,0x9Fd761F6b52298D0
   cDC2E92e01A4c5dd77E4c8f6,how are u ?,1733004408
```

Figure 3. Screenshot of interaction possibilities with the smart contract from Remix IDE

3.2 Front-end development and deployment

For the development of the decentralized messaging user interface that communicates with the smart contract, the technologies used are **Vite** and **React**, which ensure ease of development and good performance. **Vite** allows us to keep only the files necessary and optimized for production when the project is built, and **React** allows us to break down our interface into different components. In the case of this application, there is a component for displaying conversations with users, another for displaying messages and one for user input. We also need to add libraries to communicate with our wallet and smart contract. The **ether.js** library is used to manage interaction with the blockchain, while the **Web3.js** library is used to manage connections with the wallet and transactions when a message is sent. We will not go in depth to the front-end implementation because it is not the aim of the article.



```
export const CONTRACT_ADDRESS = "0xE88320031C79eaeC940d0768f428C2ab6D61842F";

export const CONTRACT_ABI = [
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "_to",
                "type": "address"
            },
            {
                "internalType": "string",
                "name": "_content",
                "type": "string"
            }
        ],
        "name": "sendMessage",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
```

Figure 4. Extract of the content of the file contract.js

We also need an important file in our front-end, “contract.js”. This file will act as a bridge between the front-end and the deployed smart contract (see Figure 4). This file contains the address of the smart contract used, as well as the contract’s Application Binary Interface (ABI), which describes its structure and data.

The final step in making the project work is to deploy the front-end. For this, any hosting provider will work, but for for this project, Amazon Web Services is used for its simplicity.

3.3 Demo Workflow

In this part, we will look at how to render the project's user interface.

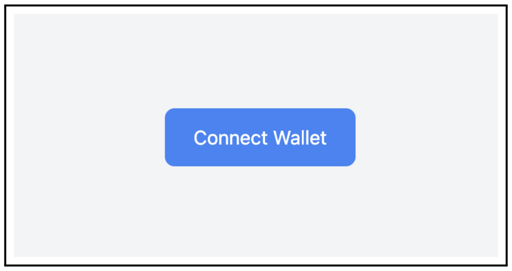
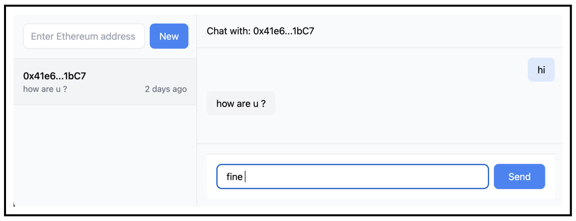


Figure 5. Wallet connection button on the dApp

Figure 5 shows us what is displayed on arrival at the site. The user must connect to his Wallet to access it, without having to create an account or enter any information.



Chat with: 0x41e6...1bC7

0x41e6...1bC7
how are u ? 2 days ago

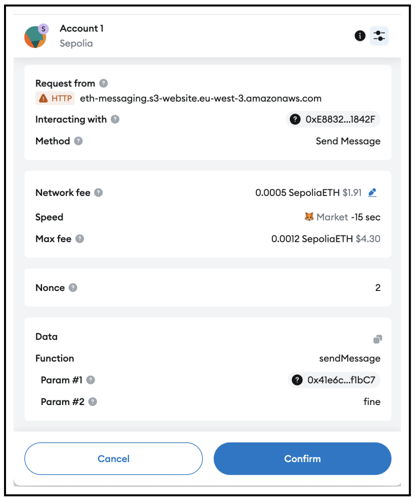
hi

how are u ?

fine

Figure 6. User interface on the dApp

As shown on Figure 6, he can then access his existing conversations or create a new one by entering an ethereum address in the top right-hand corner of the screen.



Account 1
Sepolia

Request from
HTTP eth-messaging.s3-website.eu-west-3.amazonaws.com
Interacting with 0xE8832...1842F
Method Send Message

Network fee 0.0005 SepoliaETH $1.91
Speed Market ~15 sec
Max fee 0.0012 SepoliaETH $4.30

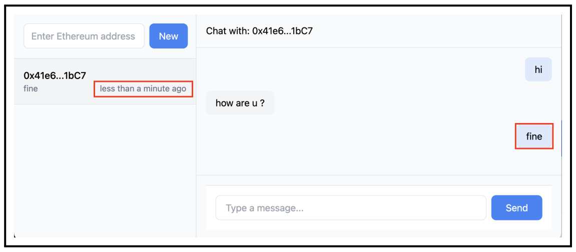
Nonce 2

Data
Function sendMessage
Param #1 0x41e6c...f1bC7
Param #2 fine

Figure 7. Transaction request to the sender

When the user wants to send a message, a transaction request will appear on the screen, confirming the settings and displaying the fees to be paid (see Figure 7).

Once the transaction has been accepted and entered into the blockchain, the user can return to the interface and see that it has been updated with the new message (see Figure 8).



Chat with: 0x41e6...1bC7

0x41e6...1bC7
fine less than a minute ago

hi

how are u ?

fine

Figure 8. User interface on the dApp updated

4. Limits of decentralized communication

Decentralized communication solves many of the problems associated with the presence of a central authority in communication systems, but it also has its inherent limitations.

The first is that it depends on a network, the blockchain in the case of on-chain, or any peer-to-peer network in the case of off-chain networks. If the network were to become unavailable or run out of users, the system would be blocked. Availability problems can also be encountered with centralized applications. So we have to ask ourselves which system is the most robust, because we know networks like Ethereum or Bitcoin are extremely robust, but we don't have enough hindsight to know about smaller ones like Polygon, which was mentioned.

The second limitation of these systems is cost, particularly for on-chain networks, as fees are required for transactions with the blockchain. Again, this is the attempt with the Polygon (Layer 2) secondary network, but it does not entirely solve the problem, as there are still fees involved.

The third limitation is the transparency of networks, which could lead to a lack of confidentiality. This may seem contradictory, but given that transactions are public and traceable, it is imperative to encrypt the content exchanged, and this must be taken into account directly in the design of messaging systems. In the case of the application demonstrated in Part 3, for example, cryptographic security would have to be implemented to encrypt the information so that it could actually be used.

Finally, and this was not mentioned in the article, there is also the technical challenge of scalability, as the on-chain and off-chain networks cannot execute an unlimited number of transactions every second. There could also be problems transmitting messages during peak usage periods.

Conclusion

Decentralized communication is a robust, secure and transparent innovation that could revolutionize the way we communicate digitally in the years to come. It offers a new perspective on the issues of confidentiality, data ownership and censorship. The interoperability enabled by decentralized networks gives users total control over their exchanges from any compatible application. Certain challenges remain, such as cost, speed and scability, but interesting prospects are emerging for the years to come, and many projects similar to those mentioned in this article are continually being developed and increasingly used (Waku, Mastodon, IPFS). After comparing on-chain and off-chain protocol models, we can conclude that they meet different needs. Off-chain protocols more easily enable instant, free communication, while on-chain protocols prioritize the robust storage of information using blockchains.

References

[1] William Metcalfe. Ethereum, smart contracts, DApps. *Blockchain and Crypt Currency*, 77:77–93, 2020.

[2] XMTP Labs. XMTP.org: Extensible Message Transport Protocol, 2024. Accessed: 2024-03-12.

[3] Jacob Eberhardt and Stefan Tai. On or off the blockchain? Insights on off-chaining computation and data. In *Service-Oriented and Cloud Computing*, ESOCC 2017, pages 3–15, September 2017.

[4] Lejun Zhang, Zhijie Zhang, Zilong Jin, Yansen Su, and Zhuzhu Wang. An approach of covert communication based on the Ethereum whisper protocol in blockchain. *International Journal of Intelligent Systems*, November 2020.

[5] Mayank Jain, Udit Takkar, Yash Gupta, and A. Tiwari. Revamping social networking using blockchain: Conceptual case-study of lens protocol. In *Proceedings of the International Conference on Innovative Computing & Communication (ICICC)*. IEEE, February 2022.

[6] Bina Ramamurthy. *Blockchain in Action*. September 2020.