How can the Carbon Footprint Issue be Solved by Blockchain Technology?

Asset Mukayev

E-mail: assetmukayev@gmail.com

Accepted for Publication: 2023
Published Date: November 2023

Abstract

Efforts to fight climate change are becoming increasingly important as greenhouse gas concentrations rise. The concept of carbon footprint, which measures the impact of human production and consumption activities on climate change, plays a significant role. This paper explores how blockchain technology can be employed to trace and reduce the carbon footprint of products and organisations. It presents a conceptualization of a blockchain-based traceability platform for carbon footprint, emphasising its potential impact on environmental sustainability.

Keywords: Blockchain, Solidity, Smart Contract, Decentralisation, Sustainability, Carbon Footprint, CO2 Emissions

1. Carbon footprint

Nowadays, most people are familiar with the term "carbon footprint" from the continuous discussion of climate change. In recent months, it has become more and more common in corporate, political, and media circles. However the exact meaning of "carbon footprint" is still unclear, creating misunderstandings about its definition, quantities, and measurement.

The conventional notion of a carbon footprint, which has its roots in the terminology of Ecological Footprinting (Wackernagel 1996), is that it is a representation of gaseous emissions from human production or consuming activities that are linked to climate change. However, there isn't much agreement on how to gauge or quantify this footprint. There is disagreement about the definitions, which range from direct CO emissions to thorough life-cycle assessments of greenhouse gas emissions.

Important queries come up: Should additional greenhouse gases, such as methane, be included in the carbon footprint in addition to carbon dioxide (CO2) emissions? Should it cover compounds without carbon, like NO, a strong greenhouse gas, in addition to carbon-based gases? Even whether the carbon footprint should only take into consideration materials that have the potential to cause global warming is up for debate. After all, although carbon-based, emissions such as carbon monoxide (CO) have an impact on the environment and human health and have the ability to convert into carbon dioxide (CO_2) through atmospheric processes. Should all emission sources be included in the metric, including emissions related to non-fossil fuels such as CO2 from soils?

The primary question that arises is whether direct, on-site emissions should be the only focus of the carbon footprint, or if indirect emissions from upstream production processes should also be included. Does it essentially capture the effects of the goods and services in question across their whole life cycle? If so, how can these effects be measured and where should the demarcation line be drawn?

1.1 Carbon Footprint Measurement

As with the notion of "Ecological Footprint," which is measured in hectares or "global hectares," the term "footprint" indicates an area-based unit of measurement (Wackernagel et al. 2005).

Carbon Footprint Measurement refers to the quantification of greenhouse gas emissions, primarily carbon dioxide (CO2), associated with human activities, products, or

organisations. It is used to assess the environmental impact in terms of climate change. Carbon footprint measurement is typically expressed in units of carbon dioxide equivalents (CO2e) and is an important part of sustainability assessments.

Example: Calculating the carbon footprint of a product, such as a smartphone, involves measuring emissions from its entire life cycle, including raw material extraction, manufacturing, transportation, usage, and disposal.

2. Methodologies for Carbon Footprint Analysis

2.1 Process Analysis

Process Analysis is an approach to calculating carbon footprints by examining the entire life cycle of a product or process. It involves identifying and quantifying emissions at each stage, from resource extraction to disposal, to understand the environmental impact comprehensively.

To calculate the carbon footprint of a loaf of bread, process analysis would consider emissions from wheat farming, transportation, milling, baking, packaging, and distribution.

2.2 Input-Output Analysis

The input-output analysis combines life cycle assessments with input-output models to assess the carbon footprint of individual products or activities. It involves analysing the interdependencies between various sectors of the economy and how they influence emissions.

Input-output analysis can be used to estimate the carbon footprint of a city by considering the economic relationships between different sectors, such as transportation, energy, and agriculture.

3. What is Blockchain?

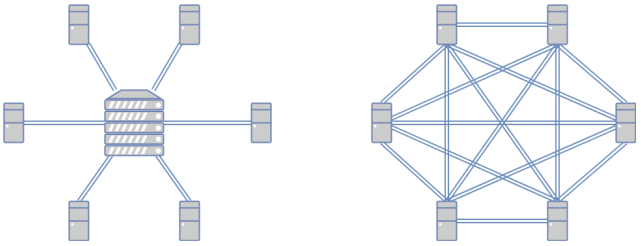
There are two definitions of the term Blockchain:

- Replicated Distributed Database;
- A continuous sequential chain of blocks containing information.

Data Replication - is the process of storing data in more than one site or node.

A distributed database - is a database in which data is stored across different physical locations.

For a better understanding of the technology, let us recall that there are two types of computer network architectures:



1. *Client-server network;*
2. *Peer-to-peer (P2P) network.*

The organisation of the network in the first way implies centralised control of everything: applications, data, and access. All system logic and information are hidden inside the server, which reduces the performance requirements of client devices and ensures high data processing speed. It is this method that has become most widespread these days.
Peer-to-peer or decentralised networks do not have a main device, and all participants mostly have equal rights. In such a model, each user is not only a consumer but also becomes a service provider himself.

An early version of peer-to-peer networks is the USENET distributed messaging system, developed in 1979.

3.1 Illustration

Let's look at a simple example of how blockchain technology works without computers.

Suppose we have a group of 10 people who want to perform currency exchange operations outside the banking system. Consider the actions performed by participants in a system where ordinary sheets of paper will represent the blockchain:

Each of the participants has a box in which he will add sheets with information about all transactions made in the system.

Each participant sits with a piece of paper and a pen and is ready to record all transactions that will be made.

If Participant No. 1 wants to send 100 USD to Participant No. 3, then he announces this to all participants.

All participants check whether Participant No. 1 has a balance sufficient to complete the operation. If so, everyone makes a note of the transaction on their sheets.

After that, the transaction is considered completed.

Other participants also perform exchange operations. Participants continue to announce and record each of the transactions carried out. When the sheet is filled, it must be put in a box and taken as a new one.

The fact of placing a sheet in a box means that all participants agree with the validity of all operations performed and that it is impossible to change the sheet in the future. This is what ensures the integrity of all transactions between participants who do not trust each other.
To avoid collision between the participants, adversarial models are used. The integrity of all operations in the new block implies the use of user assets as collateral (Proof of ownership Model), or a complex computational task (Proof of work Model).

3.2 Stages of Block Formation

There are several main stages of block formation:

The first step is the definition of a transaction. The sender creates a transaction that contains information about the recipient's address, the subject of the transaction (amount of funds, goods, etc.), and a cryptographic digital signature confirming the authenticity of the transaction and its legality. Network nodes are notified about the transaction and verify its authenticity by decrypting the electronic signature. If the transaction passes verification, then it goes into standby mode for inclusion in the block.

Creating a block. Blocks containing information about transactions are linked cryptographically and chronologically in a "chain" using complex mathematical algorithms. New blocks are always added strictly to the end of the chain. One of the network nodes collects transactions that are in standby mode once for a certain time interval, forms a block from them and sends it for confirmation to other network participants for verification and joining the chain.

Block validation. Nodes responsible for validating blocks receive a request to check the created block. They start a repetitive process that requires approval from other operator nodes in order to recognize the block as valid. The encryption process, known as hashing, is performed by a large number of different computers running on the same network. If, as a result of their calculations, they all get the same result, then a unique digital signature (signature) is assigned to the block.

Joining the block to the chain. When all transactions in a block are approved, the new block becomes attached to the common chain. As soon as the registry is updated and a new block is formed, it can no longer be changed. Thus, it is impossible to fake it. You can only add new entries to it.

It is important to take into account that the registry is updated on all computers on the network at the same time.

4. Role of Blockchain in Carbon Footprint Management

4.1 Tracing Carbon Footprint

It is not easy to find articles alluding to the carbon footprint traceability of a product in the literature. Some articles present methods for calculating carbon emissions, as is the case of (Li et al., 2013). Li et al. present a method for tracing the carbon flow to determine carbon emissions obligation from electricity consumption. In (Cordero, 2013), the author briefly surveys carbon footprint estimation approaches within supply chains.

Fu et al. propose a framework to expose the FAMI (fashion apparel manufacturing industry) carbon emissions to the general public and define a set of guidelines for reducing carbon emissions at all stages of manufacturing (Fu et al., 2018). The study starts by reviewing existing blockchain applications to improve sustainability and then the authors propose a blockchain-based solution to encourage lower carbon emissions (Fu et al., 2018).

In (Pan et al., 2018) a blockchain-based application for carbon trading is proposed.
To our knowledge, there is not in the literature a proposal presenting a platform for tracing a product's carbon footprint using blockchain technology. Despite this, the next subsection presents some works that use the blockchain technology for value chain operational support or location traceability.

3.2 Use of Blockchain in Sustainability

Blockchain is increasingly employed in various sectors, including:

Clear and Unchangeable Record-Keeping: Blockchain technology offers a tamper-proof, transparent ledger for tracking carbon emissions and reductions. This guarantees that data cannot be changed or removed once it has been

recorded, which is essential for preserving the accuracy of carbon footprint data.

Traceability of Emissions: Using the blockchain, any carbon emissions event, like energy use or transportation, can be documented as a transaction. These transactions provide thorough traceability of the sources of emissions since they may be connected to particular actions, goods, or services.

Emission Reductions with Smart Contracts: Automating the verification and validation of emission reductions is possible using smart contracts, which are self-executing agreements with pre-established rules. For instance, a smart contract can cause the issuing of carbon credits when an organisation lowers its emissions.

Carbon Credit Management: Efficient carbon credit issuance, trading, and retirement are possible using blockchain technology. This facilitates businesses' efforts to invest in carbon reduction projects in order to offset their emissions. Because of blockchain's transparency, carbon credits are guaranteed to be real and unaffected by double counting.

Supply Chain Monitoring: The manufacturing and delivery of commodities are major contributors to carbon emissions. Blockchain technology can be used to track a product's carbon footprint along its entire supply chain, enabling businesses and customers to make well-informed decisions regarding the environmental impact of their purchases.

Blockchain technology has the potential to involve stakeholders in efforts to reduce carbon emissions, including consumers. Businesses can enable consumers to make decisions based on a product's carbon footprint by granting access to their emission data on the blockchain.

Decrease in Administrative Expenses: Conventional methods for carbon accounting and verification can be expensive and time-consuming. By streamlining these procedures, blockchain lowers administrative burden and might increase the financial viability of carbon offset schemes.

5. Smart Contract Implementation

5.1 Solidity Smart Contract

Solidity is a high-level programming language used for implementing smart contracts on several blockchain platforms, including in Ethereum (Bragagnolo et al., 2018). It is an object-oriented programming language whose syntax is similar to JavaScript.

Beyond simple data types, Solidity allows for the creation of complex data structures that can include value mappings and contract inheritance. State variables of a contract are kept on file in contract storage, which is why they are kept on the Ethereum blockchain.

Following publication, Solidity contracts will operate on the Ethereum Virtual Machine (EVM) and can be called using their blockchain addresses. For the domain model that was previously described to be implemented with solidity, every entity must be defined as a structure (struct). Here is a list of the principal structures:

```
pragma solidity >=0.4.2;
contract CarbonFootPrint {
    //-- Structures - Entities Implementation --
    struct Product{
    uint32 id;
    string name;
    string description;
    bool intermediate;
    uint32 idOrganization; //ref to Org.
    uint32 idUnit; //reference to Unit
    uint32[] productFootPrints;
    //refs to ProductFootprint
}
struct ProductFootprint{
    uint32 id;
    uint32 co2eq; // base
    uint16 exp; // exponent
    uint32 idProduct;
    uint32 year;
    string month;
    uint32 idMA; //ref to MonthlyActivity
}
struct MonthlyActivity{
    uint32 id;
    string description;
    uint32 co2e;
    uint16 exp;
    string month;
    uint32[] productQuantities;
    //refs to input prod quantities
    uint32 output; //ref to output Prod FootPrint
    uint32 finalProductQty;
    uint32 idOrganization;
    uint32 idUnit;
    uint32 idYear;
    address idUser;
    unit32[] productionCosts;
}
struct MonthlyFixedCost{
```

```
    uint32 id;
    uint32 co2e;
    uint16 exp;
    string description;
    uint32 quantity;
    string month;
    uint32 idCostType;
    uint32 idOrganization;
    uint32 idYear;
}
```

Unsigned integers (uint16, uint32) have been used for every numeric type, because Solidity currently does not support signed integers (int). It also does not support floating point numbers, such as float or double. This way, every decimal value is implemented through a base integer value and an exponent. See, for instance, the CO2e attribute in the Product Footprint Entity, which contains a product's carbon footprint CO2e value, and is implemented through uint32 co2eq; uint16 exp; in the corresponding contract struct. The contract's state variables are defined through mappings, which allow the recording of data collections in the blockchain.

```
mapping(uint32 => Product) public products;
uint32 public productsCount;

mapping(uint32 => MonthlyActivity)
    public mactivities;
uint32 public mactivitiesCount;

mapping(uint32 => Organization)
    public organizations;
uint32 public organizationsCount;

mapping(uint32 => MonthlyFixCost)
    public mfixcosts;
uint32 public mfixcostsCount;

mapping(uint32 => Unit) public units;
uint32 public unitsCount;

mapping(uint32 => ProductCost)
    public productCosts;
uint32 public productCostsCount;

mapping(uint32 => CostType)
    public costsTypes;
uint32 public costsTypesCount;

mapping(uint32 => ProductFootprint)
    public productFootPrints;
uint32 public pfootPrintCount;

mapping(uint32 => ProductQuantity)
    public productsQuantities;
uint32 public productsQuantitiesCount;
```

An Event may be defined in the smart contract for being emitted, or fired, by some function. Fired events are stored in logs by Ethereum and may be captured by the distributed application code interfacing with the smart contract for triggering some action at the user interface level.

```
    event registUserEvent (address indexed _candidateAddress);
```

When the contract is issued, it is created through a constructor, much like objects and classes. We defined the following constructor for the CarbonFootPrint contract, so that the contract creating user is registered as the first user (admin), and a set of units is created on the units mapping (corresponding to the entity Unit), and year 2019 is stored as the first year for monthly activities and fixed costs registration:

```
//- CONSTRUCTOR AND DEFAULT VALUES SETTING
    constructor () public{
    users[msg.sender] = User(msg.sender, 0, true);
    arrayUsers.push(msg.sender);

// -- Initalize units
    addUnit("tonne", "t", 10, 0, 1, false);
    addUnit("kilogram", "kg", 10, 3, 1, true);
    addUnit("gram", "g", 10, 6, 1, true);
    addUnit("milligram", "mg", 10, 9, 1, true);
    }
```

A particularity of this type of contracts is that their deployment always has costs associated with being implemented in Ethereum. Whenever a query operation is performed, a certain amount of gas/ether is deducted from the requesting user's account. There are, however, some mechanisms that must be implemented so that, if an error occurs, that user is not discounted the full amount for something that may not be effective in the blockchain. The contract has functions for managing allowed users, such as verifying if a user is registered for using the contract and creating a new user (addUser):

```
function addUser (address _userResp,
                address _user, uint16 _tipo,
                uint32 _organization) public {
    require(
                users[_user].user_add == address(0),
                        "User already registered"
    );
    if(_tipo == 0 || _tipo == 1){
                require(users[_userResp].tipo == 0,
                        "You need admin permissions");
    }else if(_tipo == 2){
                require(users[_userResp].tipo == 1,
                        "You need organization admin
                                                permissions");
    }
    users[_user] = User(_user, _tipo, true);
    arrayUsers.push(_user);
    userOrganizations[_user].
                push(_organization);
    emit registUserEvent(_user);
}
```

Function require() serves for validation and, when failing, returns an opcode and effectively reverses the transaction and returns to the user the gas (the cost of the operation) that has not yet been spent 1 . Other functions for managing users include blocking and unblocking a user account. The contract also allows for registering organizations, of which carbon footprints must be traced/monitored. And, of course, the contract has functions for registering new products:

```
// -- Add New Product Function --
    function addProduct(string memory _name,
    string memory _description,
    bool _intermediate,
    uint32 _org,
    uint32 _unit,
    uint32[] memory _footPrints) public
    {
    bool exist = false;
    require(users[msg.sender].idOrganization == _org,
            "You need to belong to the organization");
    require(organizations[_org].id != uint32(0),
            "Organization doesn't exist");

    for(uint32 i=1; i <= productsCount; i++){
            string memory name = products[i].name;
            if(keccak256(abi.encodePacked(name)) ==
                    keccak256(abi.encodePacked(_name))){
                    exist = true;
                }
    }

    require(!exist, "Product already registered");
    productsCount++;
    products[productsCount] =
            Product(productsCount, _name,
                    _description, units[_unit].initials,
                    _intermediate, _org, _unit, _footPrints);
    organizations[_org].products.push(productsCount);
    }
```

Another important function is the one that allows creating information for a product's carbon footprint, an organisation's monthly fixed cost, and a new monthly activity for producing a given output product:

```
// -- Add New Product Footprint Function --
function addFootPrintProd(uint32 _co2eq,
            uint16 _exp, uint32 _idProd,
            uint32 _year, string memory _month,
            uint32 _idMa) public {

            require(users[msg.sender].idOrganization
                    == products[_idProd].idOrganization,
                    "The product doesn't belong
                    to your organization");
            require(products[_idProd].id != uint32(0),
                    "Product doesn't exist");

            pfootPrintCount++;
            productFootPrints[pfootPrintCount] =
                    ProductFootprint(pfootPrintCount, _co2eq,
                    _exp, _idProd, _year, _month, _idMa);
            products[_idProd].productFootPrints.push(
                    pfootPrintCount);
}
```

5.2 Smart Contract Validation

For validating the contract, at the development phase, a set of test cases have been defined. Before each test case, a test contract instance is deployed on a metamask test Ethereum blockchain:

```
beforeEach('setup contract for each test',
            async function () {
                    cfootprintInstance = await
                            CarbonFootPrint.new();
                    users_count = await
                            cfootprintInstance.getUsersCount();
})
```

This section presents some example test cases. Among other test cases, we need, for instance, to ensure that the contract allows to creation of a new organisation:

```
it("allows to create new organization",
    async function() {
    var name = "Lidia e Costa";
    await cfootprintInstance.addOrganization(
            name, [], [], []);
    var organization =
            await cfootprintInstance.organizations(1);
    assert.equal(name,
            organization.name);
})
```

It is possible to create a new product:

```
    it("allows to create new product",
            async function(){
    await cfootprintInstance.addProduct(
            "Fabric",
            "10 kg of fabric",
            true, 0, 2, []);
    assert.equal(1, await
            cfootprintInstance.productsCount());
    })
```

6. Conclusion and Future Plans

This paper details my journey in conceptualizing, developing, and validating a distributed platform application utilizing the Ethereum blockchain. First and foremost, there is a plan to create a convenient platform for everyone to track the entire production chain of a product, verify its origins, and access the carbon footprint data associated with its environmental impact. This empowers them to make more informed choices when deciding which product to buy or consume, favoring those with a lower environmental impact. Additionally, organizations that produce these traced products enhance their reputation by actively contributing to environmental well-being, ultimately improving their image and sales.

I encountered some challenges, especially regarding limitations in Solidity's data structures. The Solidity version I used lacks support for signed integers and floating-point numbers, among other constraints. In my future work, I plan to develop a data analysis tool for examining the information stored in the smart contract. My goal is to answer questions about which products, and types of products, have the highest and lowest carbon footprints, as well as how these footprints evolve over time.

Furthermore, I aim to enhance the smart contract by breaking down the code into multiple contracts using Solidity's inheritance feature, reducing both the contract's length and associated transaction costs. To streamline the contract further, I intend to store traceability data within it while moving user, organization, and other data to a cloud-based database.

Acknowledgements

This research project was carried out by Asset Muyakev with the help of open-sourced resources. Many courses were taken and books read, but without outside help from qualified specialists. Again, this is only superficial information that does not seek to claim global recognition.

References

[1] Official Solidity notebook - https://docs.soliditylang.org/en/v0.6.0/
[2] Is blockchain the future of financial technology? - https://habr.com/ru/companies/selectel/articles/347848/
[3] Carbon Footprint Analysis - https://www.financestrategists.com/wealth-management/esg/carbon-footprint-analysis/
[4] Li, B., Song, Y., and Hu, Z. (2013). Carbon flow tracing method for assessment of demand side carbon emissions obligation. In TRANSACTIONS ON SUSTAINABLE ENERGY, volume 4. IEEE Computer Society
[5] Luu, L., Chu, D.-H., Olickel, H., Saxena, P., and Hobor, A. (2016). Making smart contracts smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS ’16, pages 254–269, New York, NY, USA. ACM
[6] Distributed Application (DApp) of real working model - https://github.com/xicosantos98/CarbonFootPrint
[7] Wiedmann, T. and Minx, J. (2008). A definition of ’carbon footprint’. In C. C. Pertsova, Ecological Economics Research Trends, chapter 1, pages 1–11. Nova Science Publishers, Hauppauge NY, USA
[8] Relevancy of the problem in Kazakhstan - https://tradingeconomics.com/kazakhstan/co2-emissions#:~:text=KT%20in%201999.-,CO2%20Emissions%20in%20Kazakhstan%20decreased%20to%20211207.35%20KT,from%20211896.70%20KT%20in%202020.&text=Carbon%20dioxide%20emissions%20account%20for,climate%20change%20and%20global%20warming.
[9] Confronting the Carbon-Footprint Challenge of Blockchain - https://pubs.acs.org/doi/10.1021/acs.est.2c05165