Leveraging the Escrow-Holding Abilities of Ethereum Smart Contracts to Incentivize Account Creation for the Widespread Adoption of Web Monetization Schemes

Sophia Lichterfeld[1], Garima Rastogi[2], and Kyle Hogan[3]

[1]The Winsor School, Boston, MA
[2]Virtual Learning Academy Charter School, Exeter, NH
[3]Massachusetts Institute of Technology, Cambridge, MA

Abstract

Traditional Web Monetization (WM) schemes that stream micropayments directly to the website owner throughout the time the user spends on the page have faced significant challenges in acquiring the widespread adoption of their platforms because they require full website participation to be implemented. However, many website owners are still unfamiliar with cryptocurrencies and online wallets; therefore, it becomes a major hindrance to obligate website owners to have already set up a completely functional WM system before any user can begin employing WM with the website. Our proposal addresses this barrier by providing users with the option to initiate WM on a web page even before its owner has had the chance to establish their end of the system. We introduce a scheme where any user wishing to employ WM on a site can begin streaming micropayments to a common smart contract address where the money will be temporarily held in escrow. Owners wanting to retrieve this revenue must adopt the WM standard for future use; thus, our approach ultimately aims to encourage the propagation of WM as a viable alternative to ads or subscriptions, especially for small websites.

Keywords: *Web monetization; Smart contracts; Ethereum; Escrow; Micropayments; Solidity; Digital advertising; Decentralized Finance (DeFi); Coil; World Wide Web Consortium (W3C); Oracle*

Acknowledgements

We would like to sincerely thank our mentor, Kyle Hogan, for continuously supporting and mentoring us throughout this project as well as the MIT PRIMES program for providing us with the opportunity to pursue this research.

Author contributions
Project Concept and Design: K.H., S.L., G.R.; Programming and Implementation: S.L.; Writing of Manuscript: S.L.; Review and Editing of Manuscript: K.H.; Project Supervision: K.H.

The authors declare no conflicts of interest.

Date: January 16, 2024

Contents

1 Introduction **3**

2 Background and Existing Investigations into WM **3**
2.1 Ethereum Blockchain . . . 3
2.1.1 The Power of Decentralization . . . 4
2.1.2 Smart Contracts . . . 4
2.2 Previous Trials with WM . . . 5
2.2.1 Coil and Other Attempts at WM . . . 5
2.2.2 W3C's WM API . . . 5

3 An Approach to Encouraging Utilization of the WM API Standard **6**
3.1 Leveraging Escrow in Smart Contracts . . . 7
3.1.1 Proposed Scheme Outline . . . 7
3.1.2 A User's Perspective . . . 8
3.2 Coding Environment and Dependencies . . . 8

4 Implementation of an Algorithm Creating a Multi-Send Smart Contract with Retrievable Escrow **8**
4.1 Escrow collection in a Single Smart Contract . . . 9
4.2 Browser Input for Retrieving Money . . . 10
4.3 Integration into WM API . . . 10

5 Conclusions **11**

6 Discussions **11**
6.1 Expiration Dates . . . 11
6.2 Off-Chain Channels for Scalability . . . 12
6.2.1 Quantitative Analysis of Transaction Fees . . . 12
6.2.2 Contract Fees Hindering Feasibility . . . 13
6.2.3 Layer 2 Solutions to Micropayment and Contract Fees . . . 13
6.3 Potential of the Ethereum Name Service . . . 14
6.4 Ethical Considerations . . . 14
6.4.1 Environmental Impacts and Proof of Stake Consensus . . . 14
6.4.2 Balancing Privacy and Transparency on the Blockchain . . . 15
6.5 Impact of Novel WM Owner Account Creation Scheme . . . 16

1 Introduction

In 2020, website owners in the United States lost an estimated 12,120 million dollars in profits due to ad blocking [33]. Currently, small businesses are funded primarily with ads as subscription models have proven challenging to set up and difficult to manage for users and owners themselves alike [43]. However, ads are unwelcome by many as they distract from or delay access to the solicited and intended content. Furthermore, some are alarmed that data collected and shared with companies to show more tailored or suitable ads infringe upon their personal privacy. These concerns raise the question of whether there are alternative systems of supporting small content creators monetarily without hindering one's enjoyment of their content.

One potential avenue consumers have previously explored to pay for the content they consume other than by viewing ads is that of streaming micropayments, transfers of amounts of money worth fractions of a cent, throughout the time the user views the webpage. Cryptocurrencies, such as Bitcoin (BTC) and Ether (ETH), are particularly well suited for micropayment transactions because they can be split into minuscule fractional values rather than being confined to incrementing cent by cent (e.g. one could produce the equivalent of 0.0023 USD with a cryptocurrency, an amount that could not be shown with conventional coins) [1]. Supporting content creators by gathering micropayments would be particularly beneficial to smaller businesses and push for decentralization rather than facilitating the domination of the digital market by large enterprises such as Amazon or Facebook [35].

Applications of micropayments have been attempted for a wide variety of use cases and yielded varying degrees of success. For instance, Blendle [6], which gathered news articles from many renowned media outlets on its app and allowed users to pay under a dollar to see each article, did not gain any profit in five years; at this point, recognizing that micropayments appeared to be an infeasible business model, it transitioned to a subscription-based system. Likewise, SatoshiPay [52], a button that appeared on websites and allowed users to send content creators small amounts of money the user had previously loaded into a digital wallet, found that few sites were willing to implement their scheme, often out of perplexity for its functionality or out of difficulties in accessing cryptocurrencies. Other particularly noteworthy examples of attempted micropayment applications include Coinqvest [46], the Raiden Network [20], and Roll [58].

Clearly, the widespread adoption of micropayments still faces multiple challenges. Most notably, transaction fees are ultimately so high that, when taking into account the minuscule amount of money being sent, the fee, in proportion, seems unreasonable [14]. Some groups have tested holding several micropayment transactions in a third-party digital wallet so that they could all be officially sent at the same time with just one fee. Yet, in these cases, power ended up being predominantly centralized in the hands of the intermediary, and it was difficult to determine a third-party entity that was sufficiently trusted across the board [15]. In addition, micropayment processing can be slow or delayed, and consumers are hesitant to make a large volume of transactions, even several little ones, partly since they lack familiarity with digital infrastructure and currencies [48].

At this point, one of the main challenges hindering the widespread adoption of micropayment schemes to fund web content is that both users and website owners must have set up all necessary features to participate in the procedure before it can be put into use in any way. However, establishing the required elements may take time and expertise that is not immediately available to small content creators or may not initially appear as a worthy investment. Hence, we approach this issue by designing a scheme where only users must have all components prepared beforehand to implement a micropayment-based alternative to ads; the money they transact will be held securely in escrow for the website owner to collect at a later date when they have had sufficient opportunity to install a functional digital wallet system that can receive the money.

2 Background and Existing Investigations into WM

There are several platforms and projects that provide precedents upon which our research builds. We targeted our protocol development toward the Ethereum blockchain, one of the largest cryptocurrency systems globally [25]. Moreover, diverse approaches to WM have already been initiated by several groups, including the World Wide Web Consortium (W3C). Specifically, we focus our investigations on amending W3C's WM API protocol, ultimately hoping to assist a shift to greater reliance on WM as opposed to ad revenue for content creators.

2.1 Ethereum Blockchain

Ethereum [18] is a program for performing transactions with its cryptocurrency, Ether (ETH), based on blockchain technology. Established in December 2013, Ethereum has become the cryptocurrency with the greatest market share after Bitcoin. However, Ethereum and Bitcoin use vastly different algorithms to ensure security, decentralization, and transparency; thus, they lend themselves to slightly differing purposes [3].

2.1.1 The Power of Decentralization

Ethereum was established with the goal of fostering a more decentralized, open transaction network. When a participant in the network requests a transaction, a block is automatically created with all the relevant data. This information is then broadcast to all computers, or nodes, on the network. These devices must approve the request, a process intended to increase the security of the system, and certain nodes receive monetary compensation for correctly validating requests as an incentive to remain truthful (see proof of stake) [67]. Finally, the block is appended to the existing blockchain to solidify all the transactions within it; at this point, the transaction cannot be changed without having to modify the blockchain history on the majority of the nodes on the network, a virtually impossible task. The use of a public blockchain facilitates transparency and security as any device can check the history of transactions and all essential information will be contained in the corresponding block [37].

Ethereum is a programmable system, and users can make their own applications (or add to existing ones) that perform various functions and tasks on the blockchain, even beyond transacting money; thereby, Ethereum claims to have become more than a simple payment method: rather, it brands itself as a marketplace for a variety of services [66]. During transactions, the two parties negotiate directly with each other as there is no entity that oversees Ethereum. Ergo, Ethereum can be considered a system of decentralized finance (DeFi) [57].

2.1.2 Smart Contracts

Solidity [53], the programming language under which Ethereum operates, is considered to be a contract-oriented language. Thus, a key building block of Ethereum is the smart contract, a section of code on the Ethereum blockchain that automatically executes without the involvement of a third party once both ends of an agreement have been upheld. In a decentralized network such as Ethereum, the ability to ensure that previously set terms are autonomously enforced without requiring another entity to superintend the interaction is a powerful mechanism that can be leveraged in numerous situations [30].

In practice, a smart contract can be thought of as an if/then statement that will perform a particular transaction in response to both parties' meeting of specified terms. The requirements written into a contract are considered to be "immutable"; they cannot be revised by any user. Given that the conditions that were originally laid down are permanently enshrined in the contract's code, the parties involved do not need to trust each other when opening a new contract. There is no room for negotiating interpretations of an agreement since the requirements and outcomes are exactly known and foreseeable to both parties from the start; in the same set of circumstances, the contract will always yield the same result. Moreover, there is little delay between the moment when the terms of the contract have been met (by both sides) and when the ends of the contract are fulfilled. Smart contracts are recorded on the public blockchain and accessible to be viewed by all nodes, so anybody can see the history of executions of a contract [32] [37]. Smart contracts have been applied in a wide variety of use cases, including CryptoKitties [12], AXA flight insurance [5], and FNZ (previously Authenteq) [28].

In a smart contract involving a financial transaction, a critical property is the contract's capability to hold money in escrow until the conditions are met by all parties; a seller should not receive a payment if they have not yet delivered their service or product, and, vice versa, the buyer should not receive the service or product if they can not provide assurance that they will compensate appropriately. With a smart contract, the buyer sets aside the necessary amount of money, and this amount, which is referred to as being kept in escrow, is now inaccessible to them for any other transactions. Thereby, the buyer has fulfilled their part of the contract.

From the seller's perspective, they now know for certain that sufficient funds have been stored in escrow but that this money will not be released to them until they have satisfied their end of the contract. The parties may also agree on a period of time within which the service must be completed; if the seller does not meet this due date, the money in escrow is released back to the buyer. Ultimately, the smart contract checks that all conditions have been met and then releases the money that the buyer had previously put into escrow to the seller [31].

While it is generally relatively easy for the smart contract to check whether the buyer's end of the contract has been upheld simply by seeing what amount of currency has been put into escrow when looking at the transaction history on the public blockchain, there can be some more complexities when it comes to verifying the conditions the seller must meet. In some cases, there may be a transfer of tokens (NFT, for example) or another sort of transaction that is similarly broadcast on the public blockchain. However, at times, data about what has occurred outside of the blockchain may be necessary to validate the seller's release conditions. In these instances, an Oracle, which reports information curated beyond the blockchain (the date a product has been delivered, for instance), can be utilized to inform the smart contract when it is appropriate to release the money in escrow to the seller [42].

This system for financial transactions is highly efficient as it eliminates the need for a third party, thereby decreasing transaction costs and time [19]. Furthermore, since the smart contract cannot be tampered with by any user on the blockchain once it has been deployed, this procedure heightens security and trust [36]. The question we discuss in this paper is in what ways smart

contracts could be incorporated into payment channels, especially by transiently acting as an interim node in their ability to hold money in escrow, to offer an alternative method for compensating content creators.

2.2 Previous Trials with WM

In recent years, many entities, seeking to transition further toward the ideal of a decentralized and open yet secure web environment, have ventured into the field WM with varying approaches and have hit a variable range of metrics of achievement. Two notable examples of platforms that have attempted to tackle WM from vastly different vantage points are Coil [11] and the WM API [61] developed by the W3C.

2.2.1 Coil and Other Attempts at WM

There have been multiple trials to replace online revenue from advertisements with direct payments for viewing content through third-party applications; nonetheless, overall, these projects tend to have had trouble gaining sufficient traction from both users and content creators. The most notable of these tests is perhaps Coil, a monthly subscription that continuously flowed micropayments to participating website owners while a user was viewing their website [11].

Coil was a novel WM scheme created in 2018 by Stefan Thomas. With Coil, users could support smaller content creators without seeing ads or giving any private user information [51]. A Coil user paid for a monthly five-dollar membership using their credit or debit card. They could then access websites online with certain browsers, such as Puma, that offered the option to install Coil as a browser extension. Thus, when users opened a web-monetized site, micropayments were sent directly from Coil to the website owner's account in proportion to the amount of time spent on the page. Coil generally sent creators 36 cents for every 60 minutes a user spent on the page. Creators who wanted to participate in this scheme could set up a digital wallet (using Uphold or GateHub, for example) and then attach an HTML tag on their website connecting their wallet. Creators did not need to pay to participate in Coil. However, the complications of having to set up and remember the authentication information for both an account on Coil and a linked Interledger-backed wallet in which the money would be stored profoundly reduced the widespread implementation of the scheme among website owners [45].

Some additional attempts at providing WM-based substitutions to advertising include the Brave browser [8], America Online [4] (which initially had viewers pay per minute of use but eventually transitioned to a monthly subscription model), Superfluid [55] (a many-to-one payment scheme that also streamed micropayments but sent all the accumulated money as one transaction to lower the relative transaction fee) [14], and Flattr [21] (a button incorporated into several social media sites, including Facebook and Twitter [22]; when pressed, it would give the content the equivalent of a "like" and simultaneously send money to the creator [44]). Many of these trials found their WM models to be financially unsustainable in the long term and did not see their scheme's general adoption.

2.2.2 W3C's WM API

W3C has also been developing a WM API standard for privately streaming micropayments from users to content creators [61]. This system could be implemented in place of subscriptions or ads with the WM API's main objective being to give web page owners a method of receiving compensation for their work without revealing any identifying information about the user or making the user repeatedly sign in to a financial account each time they want to participate. It is fundamentally not a form of e-commerce; rather, WM API is meant purely for streamed micropayments [62].

One of the main components that makes up the WM API is the Interledger Protocol, a process to transact payments across the web. While sending money within a bank or a financial system is relatively simple due to the established trust between these organizations, this trust does not exist between different institutions on a larger and global scale. Thus, in these cases, the process of transferring money can be extremely complicated and often requires high fees when going through mutually trusted intermediary agents [1].

Instead, Interledger provides an "open protocol" that supports the interoperability of separate financial systems. It can be used universally without any preexisting relationship between the two systems; it is not owned by any specific entity and is not better suited for any particular currency. Accounts that contain money are referred to as ledgers, and Interledger holds the money in escrow until both parties affirm that their payment is complete. It is especially advantageous for WM since it allows users to store the necessary payment information to then be seamlessly accessed for every future transaction. Interledger provides a major step toward a decentralized financial environment; however, it has yet to be applied on a broad scale as the economic landscape is currently still dominated by governments and private enterprises [29].

Another critical element of the WM API is payment pointers, more succinct strings in the format of URLs that contain payment

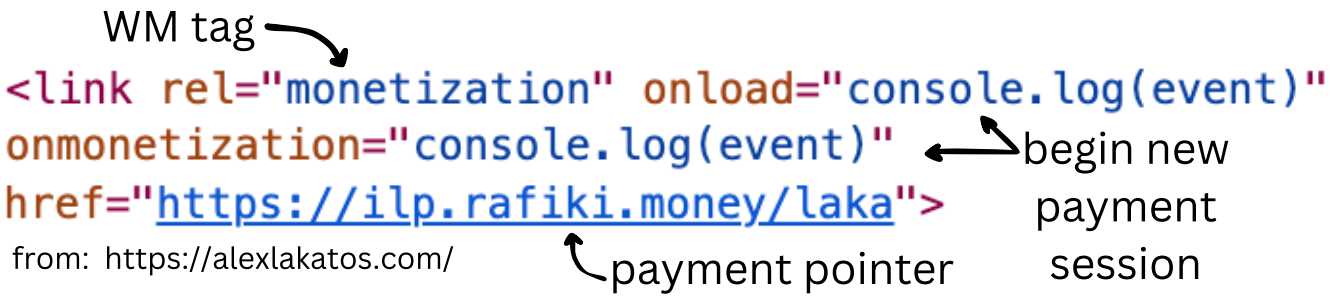

Figure 1: Web Monetization <meta> tag format

information by resolving to wallet addresses without infringing upon user privacy (Figure 1). They are easier to remember and can be securely shared. They should be placed into a meta tag in the HTML head of a web page, and, currently, they must be manually added by the website owner. The payment pointer essentially links the web page to the owner's digital wallet and can be read by the browser to establish this connection for WM use [62].

To use WM, users must create an account through a WM Provider, the entity that gives Interledger access to the person viewing the web-monetized content. They must also download a compatible WM Agent on their browser. On the owner's end, they must create a WM Receiver, which refers to the wallet to which the money will be sent [62]. Both the Receiver and the Provider go through the Interledger network, so they do not have to be connected to each other in any other fashion [29].

A page implementing WM must include a <meta> tag with the payment pointer in its HTML head to indicate where money should be sent (likely to the WM Receiver). The user's browser, which should have an internal WM Agent installed, can compute the streaming rate, the speed at which micropayments should be sent. For every payment session, the browser generates a distinct session ID and then, given the webpage's payment pointer URL, creates a unique destination address based on a shared secret for this payment session. Optionally, an independent payment receipt verifier can check and authorize the transaction by sending a Receipt Secret and Receipt Nonce to confirm that the receipts are, in fact, sent to the WM Receiver. After having established this connection to the Receiver, while the user is active on the page, the browser will count time and make continuous payments at the determined rate. The transactions are relayed from the Provider to the Receiver, and the browser communicates to the website that the payment has been completed. Throughout this process, the browser acts as a mediator so that the transaction remains private and cannot be traced back to the user [62].

Currently, WM is a browser extension, although it hopes to be incorporated directly into browsers in the future. The browser plays an extensive role in the WM API [34]; therefore, it can be valuable to briefly discuss and investigate its tasks in some greater detail. If the browser is able to identify the payment pointer in the <meta> tag of the website, it creates a new Universal Unique Identifier (UUID) that is used as the session ID for the current site visit, which constitutes one payment session between the user and the page owner. The browser requests that its WM Agent compute the rate of money flow to the page. It also recognizes the payment pointer and queries for a distinct Interledger address (referred to as the destination address) and a shared secret from the endpoint the payment pointer leads to. At this point, the browser has all the necessary components to start processing payments through the WM Provider. The browser communicates these components with the user's WM Provider to instantiate a new PaymentRequestEvent; this action also switches the browser's monetization state variable from "pending" to "started." The payment session continues at the predetermined rate until the user terminates their viewing of the site. At that point the browser's state is switched back to "pending" and the flow of micropayments is interrupted [62].

3 An Approach to Encouraging Utilization of the WM API Standard

Several factors have hindered the widespread application of WM; among those, arguably the most profound obstacle is the general lack of familiarity with cryptocurrencies and the resulting complications on the side of the website owner to implement this scheme effortlessly and effectively [1]. Taking into account this significant hurdle, we design our approach to allow users to launch WM even with websites that have yet to adopt the full protocol. While the current WM API requires website participation from the start, we give website owners a grace period in which users can begin (with the WM API) streaming micropayments that website owners can then collect after-the-fact once they have set up the necessary components to be compliant with the full WM standard. Thereby, we further provide a financial incentive for website owners, wishing to retrieve the money that has been set aside for them, to take the step of establishing their end of the W3C's WM API system.

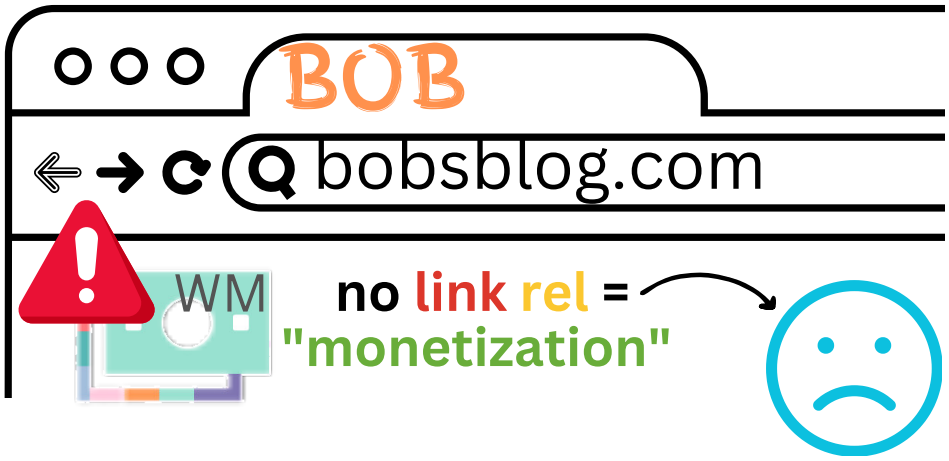

Figure 2: Current User Side — WM cannot be utilized without a legible payment pointer

3.1 Leveraging Escrow in Smart Contracts

We believe Ethereum's basis upon smart contract provides a distinct advantage for our proposed scheme. In particular, we focus on the ability of smart contracts to hold money in escrow until specific requirements are met. The autonomous nature of smart contract execution presents itself as an additional advantage to the implementation of this system. Consequently, we have developed a smart contract that can hold user micropayments from WM in escrow until the website owner has successfully installed a payment pointer into their website's HTML head and retrieves the amount from the smart contract.

3.1.1 Proposed Scheme Outline

We set out to apply a three-pronged approach to widely establishing a WM standard across websites: on the one hand, websites that already include a correctly formatted payment pointer in their header will be left to operate as they currently are; on the other, the information in the header could either be set up in a manner that is illegible to the browser or could be missing altogether. These two cases are more complicated to manage and have yet to be addressed by the W3C's WM API standard thus far (Figure 2). Regardless, they can be handled essentially identically as, in either instance, the browser is left with no apparently identifiable payment destination endpoint to which to send the money collected from WM.

If presented with one of the latter two cases, we propose using a smart contract to temporarily hold any money collected from WM in escrow (Figure 3). The smart contract's address acts as a default wallet address for WM until the website owner's payment address is appropriately added to the website HTML header as a payment pointer. Smart contracts are well-suited for this task because the address will perpetually remain constant and, thus, the payment destination address can be hard-coded; moreover, exceedingly limited human involvement is necessary for the smart contract to function once it has been set up.

However, keeping money in escrow in a smart contract can only be considered a transient solution based on the notion of "no account yet"; eventually, the owner must be able to retrieve the money held in the contract. As a website owner, to receive money currently in escrow, one must provide the smart contract with a wallet payment address. The smart contract will then prompt the browser to check the website's HTML header for a recognizable payment pointer. In this verification process, two conditions should be met before the money is ultimately cashed out: firstly, the website must have adopted the WM standard, appending an identifiable payment pointer integrated into its header so that a smart contract is no longer used going forward; secondly, the address indicated by the payment pointer the smart contract finds must match the one provided by the person who is asking to retrieve the money formerly held in escrow. By checking that these two addresses are the same, we can confirm that the owner of the website, the only person who can edit the website (and, thus, the only one who has the ability to add the payment pointer into the website header information), is also the one getting the money that has accumulated until then (Figure 4).

If the website had originally incorrectly set up the payment pointer information in the header rather than entirely omitting one, we simply add a supplemental initial step of notifying the owner that their current payment pointer is invalid and that revenue from WM will be collected in escrow until they provide an appropriate payment pointer.

Since the payment release condition (in order to receive the revenue that has been collecting in escrow) of our proposed smart contract system includes having integrated a correct payment pointer, our scheme offers a monetary stimulus for website owners

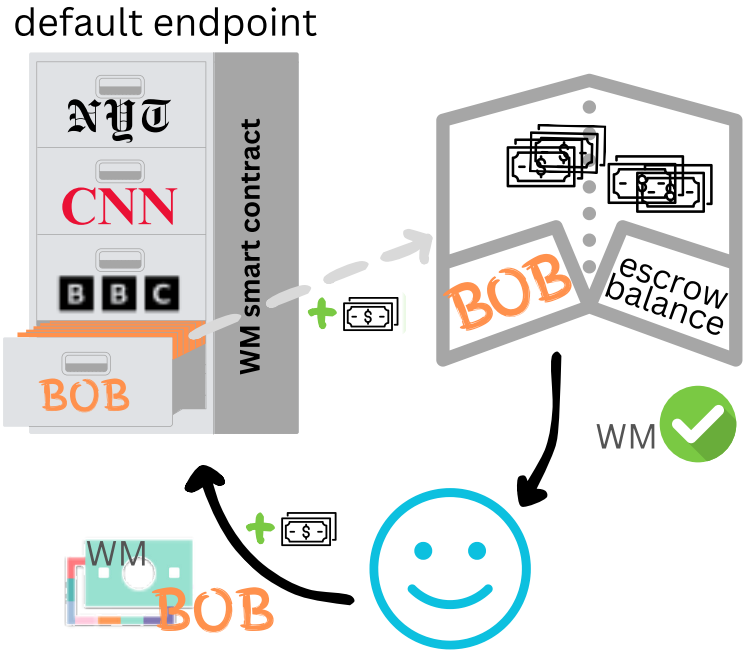

Figure 3: Proposed User Side — WM payments from users are stored in a smart contract for future escrow collection

to comply with the WM API standard and implement the existing system properly. Our WM account creation scheme is of independent interest because it can be applied separately from the remaining parts of W3C's WM API and could be used by other schemes, potentially including those similar to Coil.

3.1.2 A User's Perspective

From the user's side, there are merely two steps to integrating WM: they must join an Interledger Provider and use a browser that is compatible with the WM API (e.g. Puma, Brave, Firefox). A user who has installed a functional WM system will still have the choice between seeing ads and using WM; the owner should receive the same minimum amount regardless. However, the user could also opt for a higher streaming rate if they would like to do so. Finally, the user will not be asked to sign in anew each time they visit a website or open a page since the system will be integrated into the browser.

3.2 Coding Environment and Dependencies

We coded an implementation of the described smart contract in the Visual Studio Code IDE [60] using Solidity [53]. We complied and deployed our smart contracts for testing with Ethereum Remix [49], Truffle [59], and Ganache [23] integrated into our coding environment. For retrieving payment pointers from website HTML headers, we employed an Oracle.

When using an Oracle, we must be wary of the risks it could pose to security and anonymity as it bridges the gap between external data and the Ethereum blockchain. To ensure maintenance of the security throughout the smart contract's deployment, we utilize Chainlink [9], a decentralized platform to develop Oracles. A Chainlink node can receive data from an off-chain origin and report it back to influence the state of the smart contract [9].

4 Implementation of an Algorithm Creating a Multi-Send Smart Contract with Retrievable Escrow

Our approach features several unique components that have, to the best of our knowledge, not been combined in such a fashion previously. To initiate this scheme, we design a process by which multiple users can pay into the same smart contract, which acts in place of a wallet. This money must be held in escrow until a set expiration date, at which point the appropriate amount of

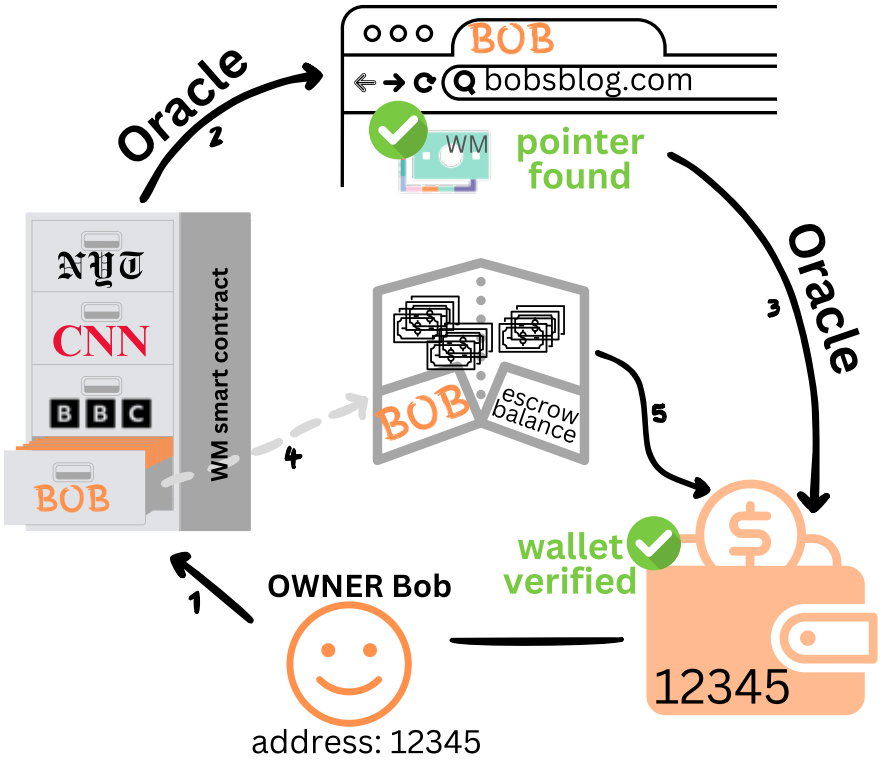

Figure 4: Owner Side — money is retrieved from escrow in the contract and paid to the owner's wallet

money will be returned to the corresponding sender if it has not yet been cashed out. Lastly, we incorporate a system by which the browser reports back to the smart contract whether all necessary steps have been taken by the owner and the smart contract verifies the validity of the provided information. Should they have appropriately adopted the WM standard, the money will be released to the owner's payment address.

4.1 Escrow collection in a Single Smart Contract

Smart contracts are typically established between a specified number of identified users. For this portion of the smart contract implementation, we were tasked with developing a method of allowing an unknown number of individuals who could not be defined in advance to have access to the smart contract. To approach this challenge, we made parts of the smart contract publically payable, specifying that ETH could be sent into the contract from any wallet address.

To preserve the anonymity of users and prevent the tracking of user behavior over time or across sites, we take inspiration from collecting money in different campaigns of a donation smart contract [40]. Our proposal stores money accumulated from user sessions for all web pages in one single, general smart contract. Each smart contract, when created, is given an address. We intend to leverage this feature of smart contracts to direct all micropayments streamed while viewing any website without proper implementation of WM to this stable address. We are, ergo, able to hard code a singular location—a default wallet address—to which money from WM should go; thus, we essentially create a substitute payment pointer address directing the WM micropayments to the smart contract. The browser can temporarily send any micropayments to this address in place of the true payment pointer to the owner's digital wallet (Figure 3).

Additionally, keeping the address the same for all user sessions makes individual transactions unidentifiable and untraceable because they are ultimately grouped together in escrow. Within this smart contract, we create distinct escrow folders for each particular website. We map the website's URL to a website ID based on a one-way hash of the root domain of the URL. By creating subsections within one aggregated smart contract, the website visited by the user is not revealed to the blockchain when the transaction showing that they paid into the smart contract is published on the public ledger.

As presented in Algorithm 1, when a user attempts to employ WM on a website that has not yet established a payment pointer, the browser directs payments to this smart contract address. The smart contract first extracts the root domain from the inputted

Algorithm 1 Escrow Collection in a Single Smart Contract

Input: $URL \leftarrow$ URL of the website
$amtStreamed \leftarrow$ micropayment value from this payment session (i.e. *msg.value*)
Notation: $rootDomain \leftarrow$ extracted root domain from URL (e.g. example.com from www.my.example.com/homepage)
$websiteID \leftarrow$ one-way hash of $rootDomain$, paired with $inEscrow$ value in *websites* mapping

extract$(URL) \rightarrow rootDomain$
hash$(rootDomain) \rightarrow websiteID$
if *websites* !contains $websiteID$ **then** ▷ escrow collection for this website does not yet exist
$websites[websiteID] \leftarrow 0$ ▷ $inEscrow = 0$ when collection for website initiated
print: new collection folder created for $websiteID$ ▷ only $websiteID$ is shared ($rootDomain$ cannot be derived)
end if
$inEscrow += amtStreamed$
transact: $websites[websiteID] \leftarrow inEscrow$
print: transaction made into WM smart contract ▷ blockchain does not see which website folder received payment

website URL and then hashes it to generate the corresponding website ID. By only performing the hash on the root domain, we ensure that there is not a new escrow folder for each page within a website that the website owner would have to cash out individually. The smart contract then checks whether a folder has already been created for this website ID, and, if not, triggers the inclusion of this website ID into the mapping with an initial escrow balance of zero. When a new website folder is created in the smart contract, an event is emitted to the public blockchain with the website ID confirming its addition. However, since we employ a one-way has to generate the website ID, the releasing of this information does not pose a significant security threat. Finally, the smart contract retrieves the amount streamed in micropayments from this user session and transacts them into the smart contract folder corresponding to this website ID. Over time, the value accumulating in escrow (mapped to each website ID separately) grows and can eventually be cashed out to the website owner (Algorithm 1).

4.2 Browser Input for Retrieving Money

The final and perhaps most critical element of our scheme is the possibility for website owners to cash out the money in escrow upon having appropriately adopted the WM standard. An interesting feature we have investigated for the payment release condition of our smart contract is the use of an Oracle, which feeds in data from outside of the blockchain [19]. In the context of the WM API scheme, we hope to have the browser check that the website's payment pointer is now legible in the HTML <meta> tag and that the payment pointer identified there matches the address provided by the individual seeking to collect the money in escrow. The smart contract's confirmation of these two parameters necessitates information about the payment pointer found in the website header (and the wallet address to which it resolves) which the Oracle can report back to the smart contract to prompt it to send out the money to the given address (Figure 4).

Upon receiving a request to cash out the smart contract, the browser, given the URL of the website, is triggered to connect to the website and attempts to identify the payment pointer in the <meta> tag of the page's HTML header. If this payment pointer has been appropriately edited or added by the website owner, the browser feeds the information into the smart contract. Thus, the smart contract confirms that the website has adopted the WM standard.

Next, the contract must verify that the individual attempting to retrieve the money in escrow is indeed the website owner. The website owner is the only person who has the ability to make changes to the website header. Hence, by validating that the browser-identified address from the payment pointer and the provided address match, the smart contract certifies that the requestor is the true owner and that the money released from escrow will go to the correct account. Thus, a second Oracle is sent to resolve the payment pointer found by the first Oracle. If the condition is upheld, having established that the correct person is attempting to retrieve the money and that the website in question has sufficiently adopted the WM standard for future direct use, the terms of the smart contract have been met, and the money in escrow under the corresponding website ID is immediately transferred to the provided wallet address (Algorithm 2).

4.3 Integration into WM API

We believe that the current API format is optimal for this scheme as it allows further changes to be made to the WM system while the main aims of the system persist. Our proposal remains compliant with the WM API standard and promotes the ultimate

Algorithm 2 Owner Verification and Money Release

Input: $addressRequest \leftarrow$ wallet address of user attempting to cash out the smart contract folder (i.e. $msg.sender$)
$URL \leftarrow$ URL of the website (should be domain of sender providing address)

Notation: $pointer \leftarrow$ payment pointer found by browser in the HTML header of the website with the given URL, via Oracle
$addressWebsite \leftarrow$ wallet address $pointer$ found on website resolves to (intended payment destination endpoint)

$pointer \leftarrow$ **Oracle**(URL) ▷ Oracle retrieves payment pointer from website HTML header
if $pointer$ is found **then** ▷ owner correctly adopted WM
  $addressWebsite \leftarrow$ **Oracle**$(pointer)$
  if $addressRequest == addressWebsite$ **then** ▷ requestor is true owner
    extract$(URL) \rightarrow rootDomain$
    hash$(rootDomain) \rightarrow websiteID$
    transact: $address \leftarrow websites[websiteID]$ ▷ money $inEscrow$ mapped for only this site is retrieved
  end if ▷ requestor not confirmed as owner; money not sent
end if ▷ $pointer$ not found or illegible, so WM standard not adopted; keep in escrow for now

objective of providing an alternative method, through streaming micropayments, to financially support online content creators without increasing the active involvement needed from either the user's or the creator's side. Thereby, we hope to encourage the WM API's usage by both users and content creators.

In addition, our proposal addresses the issue of the adoption of the WM standard by existing websites. In contrast to previous WM schemes, our approach emphasizes user and creator ease by allowing money to start accumulating in escrow in a smart contract even before a website owner has set up the standard systems necessary for WM. As owners will be required to add the correct payment pointer to their website information before being allowed to retrieve the money held in escrow, we believe our approach encourages the widespread adoption of WM in a way that previous WM schemes have not pushed for as keenly.

5 Conclusions

Identifying a primary challenge to the large-scale application of existing WM schemes to be the necessity for websites to have already appropriately set up all elements to participate in WM, in this paper, we intended to offer users an alternative to ads even before website owners have had the occasion to establish all of these components. Our novel approach leverages the ability of smart contracts to hold money in escrow in order to grant users the possibility to begin streaming micropayments into a universal smart contract even before the website owner has set up an Interledger wallet. Nonetheless, to cash out the money that has accumulated in escrow, website owners must become compliant with W3C's WM API standard. Thus, we provide an added financial motive to encourage website owners to adopt WM more broadly across the board. We believe that our proposal acts as a decisive stepping stone on the pathway toward a decentralized financial system centered around WM.

6 Discussions

There are several considerations we would like to take into account as we continue to develop this addition to WM API. On the one hand, while the popular implementation of the API would greatly support smaller businesses and protect user data privacy by replacing ads, streaming so many micropayments would prove challenging with current blockchain transaction capacities. In addition, we contemplate the incorporation of the Ethereum Name Service into WM API as an alternative method to automatically generate payment pointers for any newly created website based on its Domain Name System (DNS) registration; in this case, only existing websites would utilize our protocol, while WM API functionality would be spontaneously added for new pages. Finally, we investigate the environmental impacts of increasing blockchain activity as well as the privacy foundations and ramifications of our approach.

6.1 Expiration Dates

One component of our three-step escrow scheme has yet to be implemented is returning money to users after a set expiration period. Should a website owner not cash out the money in escrow in a reasonable period of time, we do not wish to have the

money eternally locked away from use by any party. Once the money enters into escrow, it has been set aside out of the user's wallet and is no longer accessible to them for any other purpose or transaction. Therefore, smart contracts traditionally include expiration dates. As soon as that time has been met, the money previously put in escrow is returned to the sender's wallet and is henceforth available to be used at their discretion.

However, in our model, money from all users goes into the same smart contract. This structure deeply complicates the process of sending back the appropriate fraction of the money in escrow in a specific folder to the corresponding user. The information necessary to correctly split the money in escrow back up must be preserved so that it is connected with the user who originally spent the money and so that it is sent at the respective proper time. One possibility we consider to approach this step is having the user prove to the smart contract what share of the money in escrow is theirs and when their user session occurred in order to have the money returned to them after a certain amount of time has passed. We hope to investigate the use of receipts and the amountSent function currently in the WM API scheme to accomplish this task.

6.2 Off-Chain Channels for Scalability

In the development of blockchains, when faced with the choice between decentralization, security, and scalability, the feature that most often gets sacrificed is often scalability [13]. However, should our proposal yield a large increase in the number of WM users, exploring potential scaling solutions is critical. Moreover, on the main blockchain, transaction and contract fees are often extremely high. We discuss the possibility of Layer 2 to both decrease fees and grow network capacity.

6.2.1 Quantitative Analysis of Transaction Fees

Two additional logistical concerns we are wary of regarding sending the myriad micropayments through cryptocurrencies arise when it comes to: 1) the amount of money necessary for transaction fees and 2) the level of throughput required from cryptocurrency platforms to perform the multitude of transactions.

Often, microtransactions are so minuscule that the fee for making the transaction becomes greater than the transaction amount itself; even in slightly less extreme cases, the transaction fee for micropayments remains relatively substantial and unreasonable in proportion to the amount of money transferred [39]. For Ethereum transactions, the levels of computational power necessary to perform a process on the blockchain is measured by the value of gas demanded to execute it. The total transaction fee is ultimately the product of the gas required for the transaction and the gas price. The gas limit, which represents the maximum amount that the user sets aside when intending to perform a transaction, depends on the transaction type but is typically around 21,000 units for most elementary transactions such as money transfers. Then, the gas price is found by adding the base fee, which is determined by Ethereum and fluctuates inversely with network demand, and the priority fee, an additional amount the user decides to include as a "tip" that the validator collects for their service [26].

$$gasPrice = EthereumBaseFee + userPriorityFee$$

$$totalTransactionFee = gasLimit * gasPrice$$

Ethereum blocks can be of different sizes; though a block is usually aimed to contain around 15 million gas, it can reach up to double that amount based on network popularity. Should the block size rise above this target amount, the base fee is proportionally increased for the upcoming block. Similarly, if the target amount failed to be reached in this block, the base fee for the next block is lowered to encourage more users to transact [24]. Moreover, users who set a higher priority fee, and thus must pay a higher overall gas price, are likely to have their transactions processed faster because validators have a greater incentive to include their transaction in the upcoming block [26].

One ETH is the equivalent of one billion Gwei, or Gigawei (which is equal to one billion wei) [26]. Typically, the base fee reaches lows hovering around 40 Gwei in times of very little traffic, though it can change up to 12.5% between blocks [26] [27]. Users hoping to have their transaction included in a block generally must set a priority fee greater than 2.0 Gwei [16]. On average, the Ethereum fee across all transactions is evaluated to be circa 3.64 USD [38]. As of June 2021, the minimum gas per block was 12.5 million, and, at that state when the base fee is smaller, the lowest fee for transferring money on Ethereum was 0.000567 ETH and for depositing money on Ethereum was 0.0016875 ETH [13].

Additionally, the current number of transactions that can be executed at any given time on cryptocurrency platforms appear to be profoundly inadequate to handle entirely replacing digital advertising revenue schemes with micropayment-streaming-based WM approaches. Present estimates calculate that Ethereum is capped out at 14.3 transactions per second (TPS) [13] [39]; the financial transaction company Visa, by comparison, performs approximately 1,736 TPS and has demonstrated the ability to sustain over 47,000 TPS when necessary [13].

This discrepancy exemplifies that a main impediment obstructing the expanded use of blockchain is the obligation for all devices, or nodes, on the network to stay consistent with one another. Therefore, there must be a sufficient relay time, or time gap between the formation of each new block, for virtually all the nodes on the network to be updated with the previous block's data. Ergo, there seems to be a clear-cut boundary for the length of relay time that cannot be decreased in order to ensure that each block has been appropriately broadcast across the entire network [13].

Approaching the limitation to scalability caused by relay time with increasing the block size – the number of transactions that may be included in each block – also proves ineffective as, with more transactions per block, more information, which must be shared throughout all nodes, is stored in each block. Thereby, the relay time is adjusted accordingly to a longer period, and the overall TPS fails to be swayed [13].

6.2.2 Contract Fees Hindering Feasibility

Furthermore, the fees needed to deploy a functional smart contract on Ethereum easily reach thousands of dollars [27]. The Ethereum platform estimates that the creation of a smart contract requires 32,000 gas, while any transaction included in a block demands an additional 21,000 gas [25]. Moreover, for storing every 256 bits of data, 20,000 gas is necessary [25]; thus, the greater the amount of information in the contract, the higher the gas price will be [27]. These functions make up the bulk of the required computational power, though additional gas fees for memory and initialization are also involved; however, these are relatively miniscule in comparison (see page 27 of the Ethereum yellow paper for further details) [25] [27]. Even the most succinct smart contracts demand an approximate 500 USD in gas prices alone [27]. Conveniently, for our account creation scheme, only one smart contract needs to be formed within which the escrow from all websites is stored under separate website IDs; nonetheless, this extraordinarily high value must be acknowledged.

Taking into consideration the numerous challenges concerning the scalability of blockchain technology, users have developed several systems in an attempt to combat this issue, including the use of off-chain Layer 2 channels [13].

6.2.3 Layer 2 Solutions to Micropayment and Contract Fees

One of the most pervasive forms of handling scalability is referred to as "Layer 2," which allows for multiple transactions to be executed between previously specified users off-chain rather than on the core Ethereum MainNet chain. Although this attempt to alleviate obstacles in scalability transiently comes at the expense of security and decentralization, once the off-chain is rejoined with MainNet, these concerns can be assuaged [13]. Layer 2 channels could both increase throughput to facilitate the large-scale implementation of WM schemes as well as limit fees since these channels only require two on-chain, fee-involving transactions – one to open the channel and one to close [35]. When the Layer 2 chain is closed, it merely provides a summary of the actions performed on the off-chain. Layer 2 solutions are particularly advantageous for transacting micropayments because they greatly reduce transaction fees [13]. We believe that these types of channels could be integrated with smart contracts to ensure the efficiency and consistency of our proposal.

Layer 2 channels work in parallel with the main channel, effectively increasing the throughput of the network. While on an off-chain channel, the security and decentralization tradeoff comes with the benefit of significantly lowering the amount of time and fees for performing a myriad of transactions between connected devices. In the end, the fee for a transaction comes down to an average of merely 0.049 USD on Ethereum, significantly lower than fees on Mainnet. In effect, a Layer 2 channel between several nodes acts as a grouping of multiple transactions that are completed together with one fee. The single transaction fee is split between all those who contributed to a transaction while off-chain. To move money to be dedicated to Layer 2 transactions, an account holder can either make a smart contract that will transfer a portion of their existing ETH funds (in a process referred to as bridging) or directly supply fiat money — not ETH or other cryptocurrencies — into Layer 2 [39].

State channels are one prevailing approach to Layer 2 scalability solutions. They connect to Ethereum Mainnet through a smart contract initialized when the channel is opened. The contract includes the money put into the channel, checks the approval of state updates, and mediates disputes among members of the channel. When opening a channel, users must set aside a portion of their money as a budget that is approved by all nodes on the MainNet. This deposit allows participants to transfer any amount less than that sum more fluidly while ensuring that no user tries to utilize more money than they have. Participants can then make as many feeless transactions as they wish while the channel is open before reporting the final states to the main blockchain. Transactions and state changes are performed locally, so on-chain activity is minimized. Since there are fewer nodes that must be updated and validate a transaction, the relay time is profoundly shortened, so transactions can be executed nearly instantaneously. State updates in these channels are nonetheless considered to be decisive and valid given that all parties on the channel recognized and verified them [54].

To close the channel, the participants publish the final state back on-chain. A channel cannot be closed until all users agree on

a final state or until the smart contract has expired, at which point the state is reverted to the most recent one all users in the channel have acknowledged as correct. To enhance security to the same degree as that found on Ethereum Mainnet, all nodes on-chain must approve the final state in order to successfully close the channel and grant ultimate finality to the transaction that occurred off-chain. The smart contract then confirms that every party has approved the state update and conclusively locks the final state and sends the funds to the participants accordingly [54].

One notable example of another off-chain scalability solution that has been widely implemented on Ethereum is the Raiden Network [20]. It attempts to limit the amount of work that must be performed by the main chain by allowing peer-to-peer transactions to be updated internally within the channel while only the initial and final states to open and close the channel are declared on Ethereum Mainnet. Currently, the on-chain transaction fee is approximately 40 Gwei per byte of data for both opening and closing the channel. These two Mainnet transactions are the only ones for which fees are necessary; any transaction performed on the Raiden Network does not include an additional payment if it is performed between two nodes with a direct channel between them [13].

However, when routing payments from nodes without an existing channel between them through an intermediary node, a fee may be necessary to compensate or incentivize the work of the middleman. Nevertheless, these fees are consistently significantly lower than those on the main chain. Thus, users often chose to route transactions through several intermediary nodes rather than setting up a direct channel with the recipient or sending the money on the main network since both of these options require events on-chain. In theory, with the Raiden Network, the TPS could be greater than 10,000, permitting billions of transactions to be successfully executed every day [13].

With Layer 2 scaling options, only two on-chain transactions are necessary — one to open and one to close channels — and any number of transactions can be performed off-chain between the predefined participants; thus, the transaction efficiency is increased while fees are reduced. Layer 2 solutions could be constructively applied to our proposed scheme to curtail the amount of fees to open and maintain a smart contract. We believe that Layer 2 provides an untapped resource that WM could explore to facilitate the expansion of a micropayment-based scheme.

6.3 Potential of the Ethereum Name Service

Another system we believe could be incorporated into the WM protocol to facilitate the creation of accounts to which payments could be made is the Ethereum Name Service (ENS) [17]. ENS is Ethereum's version of the Domain Name System (DNS) which links website names to IP addresses [10]. Similarly, ENS returns the associated wallet address when provided a wallet name [50]. As all websites are automatically entered into DNS upon creation [10], we would like to explore creating a link between DNS and ENS that would allow for a payment pointer for each new website to be spontaneously established so that website owners do not have the additional task of doing so themselves. ENS can already be integrated with DNS names that are under the same owner; however, owners must pursue a lengthy series of steps to connect DNS and ENS [7]. Creating an automated system to link the two upon website foundation would allow WM to be implemented easily across all newly opened sites since the link to ENS would expedite the looking up of information for an appropriate payment pointer.

Another opportunity we hope to investigate is eventually creating a registry to smooth the process of fetching and checking the payment ID. This type of documentation linking the payment address to the website or content creator could promote convenience and efficiency even as WM is more widely adopted and implemented. A potential model after which we could frame our registry is the ENS registry. ENS domain names are already stored in a registry that ties the domain to the owner. The registry also facilitates transitions between owners and includes a history of the domain creator and all of the changes the domain has undergone since then [50].

6.4 Ethical Considerations

Alarmed by the reports of the energy requirements used by cryptocurrency mining that often rival the outputs of entire nations [3], we believe that the environmental implications of our proposal must be taken into account. Furthermore, the privacy of our users is consistently one of our foremost priorities. We analyze the ways in which protecting user browsing history has been examined by previous WM trials and discuss our targeted privacy features.

6.4.1 Environmental Impacts and Proof of Stake Consensus

In 2021, Bitcoin expended over 13 gigawatts of power, which translate into upward of 65 megatons of CO_2 [3]. Due to these drastic environmental impacts, several governments, such as those of China, the European Union, and New York, have considered instating limitations or bans on cryptocurrency mining [3]. These energy requirements come primarily from the Proof-of-Work

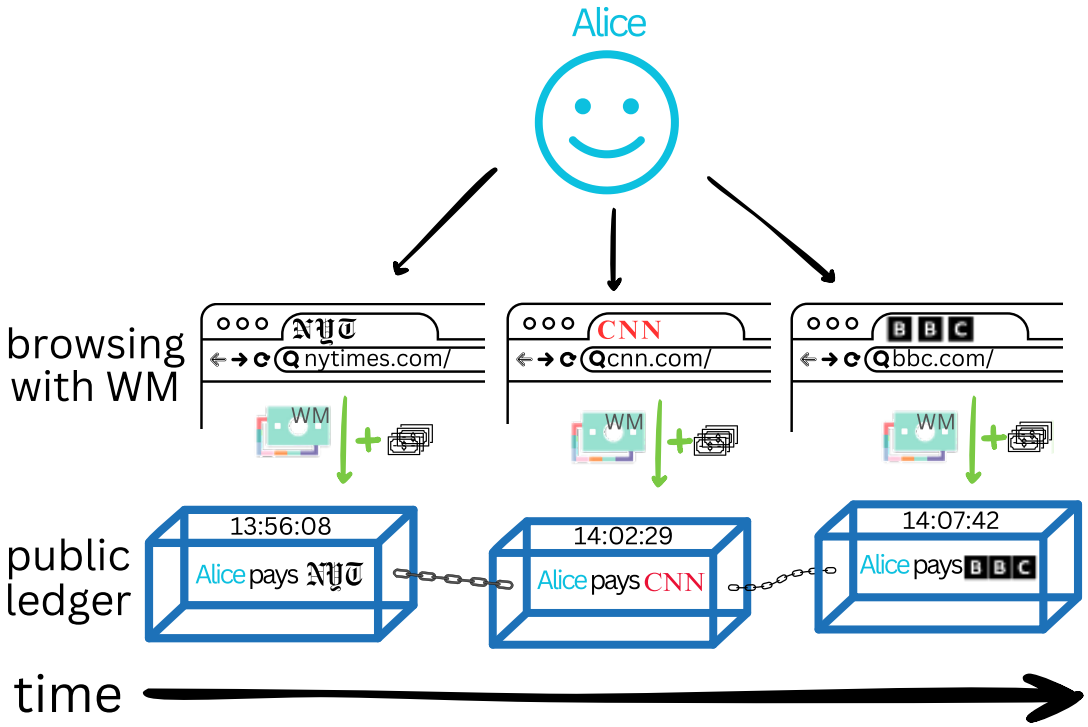

Figure 5: Deducing browsing history from WM payments on a public blockchain ledger

(PoW) consensus mechanism employed by Bitcoin. Ethereum, on the other hand, has transitioned to implementing Proof-of-Stake (PoS), an approach that profoundly reduces power expenditures and, ergo, environmentally detrimental CO_2 emissions [3].

In PoW, miners must solve a challenging mathematical problem by guessing and testing various potential answers (mining) until one of them lands on a correct solution, a process that is highly computationally expensive. The node that guessed the solution can append the block to the end of the chain and is given a monetary reward. Having greater computational power allows a device to generate and check possible answers more quickly; when Ethereum still used PoW, there were approximately 78 quadrillion (7.8×10^{16}) guesses created each day [3].

PoS, by contrast, does not depend on computationally powerful devices. Rather, each node becomes a potential "validator" by staking a certain amount of ETH, and one of these nodes is arbitrarily chosen to mint and verify the transactions in the next block. The chance of getting selected is proportional to the amount the node staked, and validators are compensated with the fees from the transactions in the block. Thus, nodes with greater wealth who can stake more are more likely to be chosen to generate the next block but do not need to exhaust significant computational power to commit to those roles [3].

On September 15, 2022, Ethereum underwent "The Merge" where its PoW consensus mechanism was replaced by a PoS one. This switch reduced its power requirements by over 99.8% [3]. Deeply conscious of the severe environmental damage as a result of cryptocurrency mining, we chose to design our scheme proposal for Ethereum, which fully employs a PoS approach, to mitigate this harm.

6.4.2 Balancing Privacy and Transparency on the Blockchain

When payments are recorded to the sites visited by a user, the transactions form a record of the user's browsing history in the blockchain. The ability to accumulate data about user browsing poses a significant concern to user privacy; thus, the potential to garner this sort of information must be diminished [62]. Coil has attempted to address possible infringements upon user data collection with a bilateral approach, implementing both "wallet-side privacy" and "sender-side privacy" [41]. Each time a user opens a new site, the wallet provider of the web page generates a new and unique encrypted destination address that is then shared with Coil so that the transaction can be completed. Coil is given information regarding the name of the wallet provider but cannot determine the specific webpage based on the encrypted address given to them by the provider. The wallet provider can decrypt the address they sent to Coil to check that there have been transactions completed from Coil to this address. At the same

time, the provider does not have any identifying information about the Coil account, so it cannot extract or track the site visiting history of a user over time [41]. This system constitutes wallet-side privacy.

However, when mediating a transaction, rather than focusing on uncovering a user's entire identity, Coil only must check their membership status [41]. Coil's sender-side privacy system is based on Privacy Pass [47]. Users initially confirm their WM utilization authorization before Coil blinds this verification. Once on a webpage, this blinded signature is provided and Coil can check its validity before launching WM on that page. In this process, the identity of the user remains unknown and unrecognizable to Coil due to the blind signature, and a user cannot be tracked over time because the blind signatures expire and are then changed within a minute [41]. Combining wallet-side and sender-side privacy, Coil ensures that neither the website browsed nor the identity of the user who accessed the site is known to them [41].

In its "good design principles" for the WM API project, W3C affirms that "privacy is key to human rights and civil liberties on the Web" [63]. One of the key goals W3C has undertaken to uphold includes preventing the sharing of user information with the WM Provider that connects the user to an Interledger wallet [64]. The API's reliance on payment pointers along with a browser-solicited shared secret to create a session-specific destination address for the WM Provider prevents the disclosure of information about which sites a user has visited [64] [65].

An issue that some have raised regarding the approach of streaming micropayments is that each individual user's payment history could generate a browsing history as well that is recorded on a public ledger (Figure 5). We closely considered two types of anonymity in relation to our proposed scheme by responding to the following questions: 1) does the person on the other side of the transaction (the owner receiving the money from WM) know the sender's identity, and 2) do the remaining users on the blockchain know the sender's identity? Neither the website owner nor the public blockchain should be able to track the identity of the sender unless a user chooses to reveal themselves. We believe we maintain both of these kinds of anonymity.

Being able to track a user across several sites or to see how many times and for how long they have visited the same site poses a significant threat to the user's privacy. However, our approach does not infringe upon the anonymity of the user; transactions by all users to any website that has not yet implemented WM are stored in escrow in the same smart contract. Thereby, information about the user's browsing sites is not shared on the public blockchain. Moreover, when the website owner receives the pooled money, it is no longer possible for them to tie any of the money held in escrow to a user's identity. Ergo, a user's behavior can also not be followed across websites over time. Hence, we ensure that user privacy and anonymity will be strictly maintained with the addition of our novel scheme to the WM API protocol. As we continue to develop our scheme, we hope to aggregate user funds before distributing them into website folders within the smart contract as an additional layer of privacy.

6.5 Impact of Novel WM Owner Account Creation Scheme

We believe that our proposal will greatly facilitate the transition from advertisement-based online revenue to a more open, decentralized digital environment that depends on WM for funding online content. We hope to join our scheme with W3C's current WM API in order to economically encourage website owners to implement WM while providing further lenience and guidance in their transition to WM. The option for users to initiate the utilization of WM before content creators have had the time or skill investment to sufficiently implement it produces a further financial impetus for owners to take the necessary steps toward adopting WM while also granting greater flexibility to users wanting to more directly and intentionally support creators rather than the large corporations behind ad revenue schemes. We hope that our scheme serves as a user- and creator-friendly avenue to the widespread implementation of WM on the global digital stage.

References

[1] A. Case, "The Current State of Micropayments and Web Monetization," *Medium* (July 22, 2021). https://caseorganic.medium.com/part-ii-the-current-state-of-micropayments-and-web-monetization-50ee2e58d332. Accessed June 28, 2023.

[2] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy Pass: Bypassing Internet Challenges Anonymously," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 164–180, Jun. 2018, doi: https://doi.org/10.1515/popets-2018-0026.

[3] A. De Vries, "Cryptocurrencies on the Road to Sustainability: Ethereum Paving the Way for Bitcoin," *Patterns*, vol. 4, no. 1, Jan. 2023, doi: https://doi.org/10.1016/j.patter.2022.100633.

[4] "AOL - News, Politics, Sports, Mail & Latest Headlines," *Yahoo* (2023). https://www.aol.com/. Accessed August 16, 2023.

[5] "AXA Travel Insurance," *AXA Partners USA S.A.* (2023). https://www.axatravelinsurance.com/. Accessed August 7, 2023.

[6] "Blendle," *Blendle* (2023). https://launch.blendle.com/. Accessed June 26, 2023.

[7] Brantly.eth, "Step-by-Step Guide to Importing a DNS Domain Name to ENS," *Medium* (August 26, 2021). https://medium.com/@brantly.eth/step-by-step-guide-to-importing-a-dns-domain-name-to-ens-d2d15feb03e8. Accessed June 30, 2023.

[8] "Brave," *Brave Software* (2023). https://brave.com/. Accessed August 16, 2023.

[9] "Chainlink Overview," *Chainlink* (2023). https://docs.chain.link/getting-started/conceptual-overview. Accessed June 30, 2023.

[10] Cloudflare, "What is DNS? | How DNS works," *Cloudflare* (2019). https://www.cloudflare.com/learning/dns/what-is-dns/. Accessed June 29, 2023.

[11] "Coil Bids Farewell, but not Goodbye," *Coil Contracts* (May 30, 2023). https://www.coil.com/. Accessed June 29, 2023.

[12] "CryptoKitties," *CryptoKitties* (2023). https://www.cryptokitties.co/. Accessed August 7, 2023.

[13] C. Sguanci, R. Spatafora, and A. M. Vergani, "Layer 2 Blockchain Scaling: A Survey," *arXiv preprint arXiv:2107.10881* (June 15, 2021). https://arxiv.org/pdf/2107.10881.pdf.

[14] D. Z. Morris, "Bitcoin's Unfinished Business: Why Micropayments Still Matter," *CoinDesk* (April 28, 2022). Accessed June 28, 2023.

[15] E. Awosika, "Can Bitcoin Fix Micropayments?," *Bitcoin Magazine* (May 9, 2022). https://bitcoinmagazine.com/business/can-bitcoin-fix-micropayments. Accessed June 26, 2023.

[16] "EIP-1559 Gas Fees: Base Fee, Priority Fee, & Max Fee," *www.blocknative.com* (August 21, 2021). https://www.blocknative.com/blog/eip-1559-fees. Accessed July 31, 2023.

[17] "Ethereum Name Service," *Ens.domains* (2023). https://ens.domains/. Accessed June 29, 2023.

[18] "Ethereum," *ethereum.org* (August 10, 2023). https://ethereum.org/en/. Accessed August 10, 2023.

[19] "Escrow," *CSIRO Australia* (June 23, 2023). https://research.csiro.au/blockchainpatterns/general-patterns/blockchain-payment-patterns/escrow-2/. Accessed June 28, 2023.

[20] "Fast, cheap, scalable token transfers for Ethereum," *Raiden.Network* (2023). https://raiden.network/. Accessed June 26, 2023.

[21] "Flattr Anti-adblock Pass," *Flattr* (July 24, 2023). https://flattr.com/. Accessed August 16, 2023.

[22] "Flattr Opens Beta of its Micropayment Platform," *Reuters* (August 12, 2010). https://www.reuters.com/article/urnidgns002570f3005978d80025777d004c8deb/flattr-opens-beta-of-its-micropayment-platform-idUS149757870320100812. Accessed June 29, 2023.

[23] "Ganache - Truffle Suite," *ConsenSys Software* (2023). https://trufflesuite.com/ganache/. Accessed August 10, 2023.

[24] "Gas and fees," *ethereum.org* (July 11, 2023). https://ethereum.org/en/developers/docs/gas/. Accessed July 30, 2023.

[25] G. Wood and others, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, Jun. 2014. https://ethereum.github.io/yellowpaper/paper.pdf.

[26] “GWEI to USD Today | Ethereum Gas to United States Dollar,” *GWEI to USD* (July 5, 2023). https://gweitousd.com/. Accessed July 31, 2023.

[27] “How Much Does Smart Contract Deployment on Ethereum Cost?,” *Antier Solutions* (May 24, 2022). https://www.antiersolutions.com/smart-contract-deployment-on-ethereum-estimated-cost-key-factors-to-consider/. Accessed July 31, 2023.

[28] "IDV & KYC," *FNZ Group* (2023). https://onboarding.fnz.com/solutions/id-verification. Accessed August 7, 2023.

[29] "Interledger Protocol (ILP): Web Monetization," *Interledger* (2023). https://interledger.org/rfcs/0028-web-monetization/. Accessed Jun. 30, 2023.

[30] "Introduction to Smart Contracts," *Ethereum* (June 23, 2023). https://ethereum.org/en/developers/docs/smart-contracts/. Accessed June 28, 2023.

[31] "Introduction to smart contracts," *Ethereum* (June 23, 2023). https://ethereum.org/en/smart-contracts/. Accessed June 28, 2023.

[32] J. Frankenfield, "What Are Smart Contracts on the Blockchain and How They Work," *Investopedia* (May 31, 2023). https://www.investopedia.com/terms/s/smart-contracts.asp. Accessed June 28, 2023.

[33] J. G. Navarro, "Cost of ad blocking in the United States from 2016 to 2020," *Statista* (January 6, 2023). https://www.statista.com/statistics/454473/ad-blocking-cost-usa/. Accessed June 29, 2023.

[34] J. Granja, "Web Monetization API: A New Web Monetization Alternative," *Medium* (March 08, 2021). https://codeburst.io/web-monetization-api-a-new-web-monetization-alternative-eb72b5c58bd6. Accessed June 29, 2023.

[35] J. Urraca, "Micropayments: Where They At?," *Revelry* (March 6, 2023). https://revelry.co/insights/blockchain/cryptocurrency-micropayments-where-they-at/. Accessed June 26, 2023.

[36] J. Wall, "An Ethereum Escrow Service That Is Easy And Safe To Use," *Invest in Blockchain* (May 30, 2019). https://www.investinblockchain.com/ethereum-escrow-service-that-is-easy-and-safe-to-use/. Accessed June 29, 2023.

[37] K. Yusov, "How Ethereum Smart Contracts Work," *Jelvix* (2023). https://jelvix.com/blog/ethereum-smart-contracts. Accessed June 26, 2023.

[38] “L2Fees.info,” *CryptoStats* (2023). https://l2fees.info/. Accessed July 31, 2023.

[39] "Layer 2," *Ethereum* (June 29, 2023). https://ethereum.org/en/layer-2/. Accessed July 29, 2023.

[40] M. Kareeem, "Create a Charity/Donation Platform on the Blockchain (part 1)," *Dev* (May 27, 2022). https://dev.to/emkay860/create-a-charitydonation-platform-on-the-blockchain-part-1-6nb. Accessed August 13, 2023.

[41] M. Sharafian, "Doubling Down on Privacy," *wrtie.as* (May 11, 2020). https://write.as/sharafian/doubling-down-on-privacy. Accessed July 29, 2023.

[42] "Oracles," *Ethereum* (June 23, 2023). https://ethereum.org/en/developers/docs/oracles/. Accessed June 28, 2023.

[43] "Overcoming 5 Disadvantages of a Subscription Business Model," *ROI CX Solutions* (2021). https://roicallcentersolutions.com/blog/overcoming-disadvantages-of-a-subscription-business-model/. Accessed June 25, 2023.

[44] P. Olson, "Flattr Goes Free To Revolutionize Online Payments," *Forbes* (April 28, 2011). https://www.forbes.com/sites/parmyolson/2011/04/28/flattr-goes-free-to-revolutionize-online-payments/?sh=65523a55528d. Accessed June 29, 2023.

[45] P. P. Koch, "How Coil Works," *DEV Community* (June 29, 2021). https://dev.to/coil/how-it-works-1b1. Accessed June 29, 2023.

[46] "Payment Processing for Cryptocurrencies and Stablecoins," *Coinqvest* (2023). https://www.coinqvest.com/. Accessed June 26, 2023.

[47] “Privacy Pass,” *Privacy Pass*. https://privacypass.github.io/. Accessed July 30, 2023.

[48] R. D. Danes, "Micropayments for Content Refuse to Take Off and Crypto Isn’t Helping (Yet)," *Defiant* (March 18, 2021). https://thedefiant.io/micropayments-for-content-refuse-to-take-off-and-crypto-isnt-helping-yet. Accessed June 26, 2023.

[49] "Remix - Ethereum IDE," *Ethereum Remix* (2023). https://remix.ethereum.org. Accessed August 10, 2023.

[50] S. Roth, "What Is the Ethereum Name Service? How It Works and What It's Used For," *Coindesk* (March 22, 2022). https://www.coindesk.com/learn/what-is-the-ethereum-name-service-how-ens-works-and-what-its-used-for/. Accessed June 29, 2023.

[51] S. Thomas, "Coil: Building a New Business Model for the Web," *Medium* (May 14, 2018). https://medium.com/coil/coil-building-a-new-business-model-for-the-web-d33124358b6. Accessed June 29, 2023.

[52] "SatoshiPay," *SatoshiPay* (2023). https://www.satoshipay.io/. Accessed June 26, 2023.

[53] Solidity Team, "Solidity," *Solidity Programming Language* (2023). https://soliditylang.org/. Accessed June 26, 2023.

[54] "State Channels," *ethereum.org* (January 19, 2023). https://ethereum.org/en/developers/docs/scaling/state-channels/. Accessed July 30, 2023.

[55] "Superfluid | Stream Money Every Second," *Superfluid Finance* (2023). https://www.superfluid.finance/. Accessed August 16, 2023.

[56] S. Weiler, "Privacy by Design," *The Interledger Community* (July 12, 2021). https://community.interledger.org/weiler/privacy-by-design-3a2k. Accessed July 30, 2023.

[57] Team Casa, "How Ethereum Smart Contracts Work," *Casa Blog* (Jan. 6, 2023). https://blog.keys.casa/how-ethereum-smart-contracts-work/. Accessed June 28, 2023.

[58] "The Future of Social Tokens Belongs to You," *TryRoll* (2023). https://tryroll.com/. Accessed June 26, 2023.

[59] "Truffle Suite," *ConsenSys Software* (2023). https://trufflesuite.com. Accessed August 10, 2023.

[60] "Visual Studio Code," *Microsoft* (2023). https://code.visualstudio.com. Accessed August 10, 2023.

[61] Web Incubator Community Group, "Web Monetization," *GitHub* (2023). https://github.com/WICG/webmonetization. Accessed June 29, 2023.

[62] "Web Monetization Explainer | Web Monetization," *WebMonetization.org* (2023). https://webmonetization.org/docs/explainer/. Accessed June 29, 2023.

[63] "Web Monetization: Good Design Principles," *www.w3.org* (2023). https://www.w3.org/projects/gftw/index. Accessed July 30, 2023.

[64] "Web Monetization: Privacy Considerations," *webmonetization.org*. https://webmonetization.org/specification/#privacy-considerations. Accessed July 30, 2023.

[65] "Web Monetization Receivers | Web Monetization," *webmonetization.org*. https://webmonetization.org/docs/receiving. Accessed July 30, 2023.

[66] "What is Ethereum?," *Ethereum* (2023). https://ethereum.org/en/what-is-ethereum/. Accessed June 28, 2023.

[67] "What is 'proof of work' or 'proof of stake'?," *Coinbase* (2023). https://www.coinbase.com/learn/crypto-basics/what-is-proof-of-work-or-proof-of-stake. Accessed June 30, 2023.