University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Deferral - High-Volume Decentralized Blockchain-Based Referral Systems

Tobias Boner
Zurich, Switzerland
Student ID: 17-707-878

Supervisor: Dr. Bruno Rodrigues, Dr. Thomas Bocek, Prof. Dr. Burkhard Stiller
Date of Submission: May 15, 2023

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

The rise of digital marketing has holistically reshaped and influenced many aspects of the marketing field and is characterized by modern technologies. During the process, existing disciplines, such as referral marketing, have been transformed to employ large automated systems dealing with high volumes of users and data representing the marketing interests of companies across diverse industries. However, most of these systems rely on centralized architectures and thus miss out on potential advantages a decentralized approach could bring. Decentralized referral systems could provide trust and transparency and tackle known issues such as complex and expensive payout processes of referral rewards.

This thesis aims to investigate and evaluate the feasibility of a high-volume decentralized referral system. The main requirements of such a system are defined in the current context, and its solution architecture is outlined. Thereby, multiple blockchain-based solution designs are developed to compare and showcase decentralized referral systems with varying complexities. The different solution prototypes are implemented as smart contracts. The smart contracts are tested and analyzed concerning their costs and performance in exemplary evaluations involving high volumes of participating users. In the best-case scenario, the final Deferral solution, including several tested and evaluated smart contracts, can serve as a framework for designing and implementing blockchain-based decentralized referral systems.

Conclusively, the examination of the generated results confirms the feasibility of a high-volume decentralized and blockchain-based referral system from a technical point of view. Furthermore, the challenges of implementing and operating such a system in a real-world environment, including the interdependence of the technical and conceptual or economical design, are discussed. Finally, the implications of varying degrees of decentralization among the different components of the Deferral solution are reviewed.

Zusammenfassung

Der Fortschritt und Aufstieg des digitalen Marketings haben weite Teile der Marketinglandschaft beeinflusst und zum Einsatz von modernen Technologien geführt. Dabei wurde in bestehenden Disziplinen wie dem Referral Marketing große automatisierte Systeme eingebaut, um mit den wachsenden Mengen von Nutzern und Daten umzugehen und die Marketinginteressen von verschiedensten Unternehmen zu vertreten. Die meisten dieser Systeme beruhen auf zentralisierten Architekturen und können so die potenziellen Vorteile, die ein dezentraler Ansatz bringen könnte, nicht nutzen. Dezentrale Systeme könnten Vertrauen und Transparenz schaffen und bekannte Probleme wie die komplexen und teuren Auszahlungsprozesse von Referral Prämien lösen.

Ziel dieser Arbeit ist es daher, die Umsetzbarkeit eines dezentralen Referral Marketing Systems zu untersuchen und zu evaluieren. Dabei werden die wichtigsten Anforderungen an ein solches System im aktuellen Kontext definiert und eine Lösungsarchitektur aufgezeigt. Darüber hinaus werden mehrere Blockchain-basierte Lösungen entwickelt und in Form von Smart-Contract-Prototypen implementiert. Die letztendliche Deferral Lösung umfasst mehrere Smart Contracts. Diese werden im Hinblick auf ihre Kosten und ihre Leistung im Umgang mit grossen Volumen von Nutzern und Daten getestet und bewertet.

Die Untersuchung der Ergebnisse bestätigt abschließend die technische Umsetzbarkeit eines solchen dezentralen und Blockchain-basierten Referral Systems. Darüber hinaus werden die Herausforderungen bei der Implementierung und dem Betrieb eines solchen Systems in einer realen Umgebung, einschließlich der Wechselbeziehung des technischen und konzeptionellen bzw. wirtschaftlichen Designs, diskutiert. Zum Schluss werden die Auswirkungen unterschiedlicher Dezentralisierungsgrade der verschiedenen Komponenten der Deferral Lösung untersucht.

Acknowledgments

The author of this thesis, Tobias Boner, would like to thank all the people who were involved in and supported this thesis. Special thanks are expressed to the supervisors of this thesis for their continuous, constructive, and helpful feedback throughout the whole process of this work.

Contents

Chapter 1

Introduction

Today's world and its modern societies are affected and shaped by the advancements of digitization more than ever before. As a logical consequence, economies and industries around the globe had and still have to adapt to increasingly digital environments and shift to more topical ways of operating. The field of sales and marketing serves as a textbook example of this paradigm change. For minor and global businesses, well-known sales and marketing techniques have been superseded by novel solutions enabled through technological progress [125]. Research has addressed the impact on sales technology and automation [1, 147]. The same applies to the marketing sector, which has evolved and changed drastically through the advancements of digitization. As a result, new means and forms of technical innovations have found their way into marketing.

Generally, marketing is known to be a dynamic and multifaceted area continuously changing and adapting to new trends and developments [110]. Many traditional aspects of marketing have transitioned to the world of digital marketing. Corresponding to the American Marketing Association *(AMA)* marketing methods or practices that utilize any form of computer can be subsumed under the term of digital marketing [8]. Essentially, digital marketing implies practices and activities supported and enabled through digital technologies and channels [103, 121]. Within this era, novel opportunities such as online platforms or ads, Social Media Marketing *(SMM)*, User-Generated Content *(UGC)*, affiliate marketing or Search Engine Optimization *(SEO)* have emerged [8, 86].

Apart from that, existing well-known disciplines such as Word of Mouth *(WOM)* or referral marketing have been modernized and transformed in line with the uptrend of digital marketing. WOM and referral marketing are long familiar to the marketing industry and were common terms before. WOM marketing which has been outlined as the personal communication between two or more people concerning a product or service [11, 118], is said to have a significant impact on people's behavior and decisions [28]. The same applies to referral or customer referral marketing, which is inherently related to WOM [145].

Referral marketing describes the interaction between existing customers and potential new customers referred to a product or service of any firm or business [145]. In connection with spreading digitization, the importance of modern referral marketing for companies and firms is increasing and expected to grow further in the future [21, 28, 72]. Like most aspects

of digital marketing, modern referral marketing is closely connected to other growing areas like SMM, UGC, and affiliate marketing, in between which the borders are often fluent. Originally, referral marketing has mainly been applied in offline settings by different firms and businesses [19]. By contrast, today's referral marketing programs and systems are digital and occur in online settings around the world by incorporating modern technologies [26, 71, 72]. Thereon, various services and platforms are more frequently engaging in referral marketing by starting and implementing referral programs and systems [60, 149].

1.1 Motivation

Digital marketing has altered the magnitude of referral programs and systems as customers participating in modern programs have substantially more reach and opportunities than was originally possible with more traditional means of communication [26, 72]. Hence, aspects including planning, designing, implementing, and sustaining a referral program have become more extensive and challenging. As a result, modern referral programs increasingly refer to large, automated systems that can encompass all these aspects. With more customers, *i.e.,* users in the case of online referral systems, more referrals can be completed, bringing new participants to the program. Consequently, more transactions must be processed, and the number of potential rewards that users can be rewarded as incentives for new referrals is growing accordingly [28, 60].

After all, the volume of transactions and data that must be stored and processed is increasing. These trends pose several challenges to modern referral marketing, but at the same time also open doors for new ways to transpose such referral programs and systems. On the one hand, the referral system and its architecture must withstand the conditions when dealing with high volumes of transactions and data. On the other hand, the overall concept and design of the referral program and system must be technically and economically reasonable to provide profitability, efficiency, and transparency to a system that must scale to large numbers of users.

The design and concept of referral programs can vary in several aspects [19, 21, 65]. The referral conditions can differ as referred customers may only have to sign up for a service or buy a product, or further steps might be required for the involved parties to complete the referral process. Concerning referral rewards, the type, the allocation between the involved parties, or the amount of the rewards can differ [68, 149]. Additionally, the point in time when rewards are distributed during the referral process can vary.

Traditionally, referral programs are implemented as centralized systems. Due to centralization, a referral system has to deal with several challenges and potential downsides. Since the number of steps, transactions, and data accumulates with a growing user base, systems can become more complex and non-transparent. Keeping referral systems simple by providing transparency [87] and trust [80] to the customers is an important factor for the success of the referral marketing of a firm [60]. To do so, the concept and structure of a referral program must be well-designed and complemented by a trustworthy and transparent architecture and implementation of a referral system. Moreover, depending on the type, *e.g.,* fiat money, and the distribution schedule, the reward payout can become slow

and expensive as complex processes, like having a bank account or credit card, could be required. During these processes, fees must be deducted for bank and credit card transactions at each step which can impact the profitability and efficiency of a system. To avoid paying fees, loyalty or referral points [21] are often distributed as rewards. Users can collect these before being used or traded against some valuable item or service. Referral points mitigate the payout problems, yet complex processes like the distribution of rewards might still be required at a later point. Nonetheless, the concept of digital referral points opens the door for new ways of first distributing the rewards to the users and then using and leveraging the rewards from there on. Referral points or tokens can be stored, freely transferred, traded, and even exchanged without any central entity opening the door for using existing Decentralized Finance *(DeFi)* ecosystems for this purpose [34, 122]. For instance, these decentralized financial systems and services could be leveraged to trade and exchange referral rewards that could come as tokens with monetary value. In other words, referral systems could be combined or integrated into the world of DeFi.

In this context, the thesis evaluates the design and feasibility of novel referral system implementation methods. The primary objective is to examine and uncover potential approaches for implementing a decentralized referral system within a solution architecture that optimizes decentralization.

1.2 Thesis Goals

The main goal of this thesis is the design and feasibility study of a decentralized high-volume referral system. Therefore, large transactions and data volumes caused by lots of participating users must be stored and transferred on decentralized systems. Yet, current tried-and-tested approaches *e.g.,* blockchain-based solutions, might reach their limits due to the large amounts of data and complex calculations as they can become expensive in these cases [108]. Hence, the focus lies on exploring potential solutions for this and related problems and implementing hands-on prototypes.

Consequently, the following goals have been defined:

- **Research on Referral Systems and Decentralized Solution Approaches:** Research on existing online and digital referral systems. Comparison of existing decentralized solutions for the feasibility of such a system. Consideration of experimental solutions and solution approaches in testing phases.
- **Conception and Design of a Solution Architecture for a Referral System:** Design of a referral program and system that can serve as a base for the technical implementation. Depiction of suitable technical solution architectures. Definition of required smart contracts and their token system.
- **Solution Prototyping and Implementation:** Implementation of prototype solutions for decentralized referral systems. Development of tests such as unit or integration tests whenever relevant to ensure the implementation's quality and correctness.

- **Evaluation and Discussion:** Evaluation of the solution, including key factors like costs or performance for the implemented prototypes. Qualitative comparison and highlighting of advantages and disadvantages of the resulting prototypes. Discussion of the feasibility of the design and usefulness of the developed solution.

1.3 Methodology

A general overview of related research or solutions is attained in the first step to accomplish the stated goals. The second step focuses on related research and concepts that serve as a comprehensive foundation for designing the referral program and constructing the solution architecture. Within this process, examples of existing and comparable referral programs are discussed.

Subsequently, the solution requirements are identified, and the solution design and architecture are examined. After, the focus shifts to technical research for potential decentralized solution approaches. From there on, the solution prototypes and their architecture are designed and eventually implemented. The design and implementation processes can complement each other, as it may be necessary to test the feasibility of certain design concepts with implemented prototypes to ensure their viability. Hence, the solution design and architecture can still be adapted during the implementation phase and vice versa. Moreover, aspects and factors of the systems that can be assessed to other comparable systems are illustrated. Upon completion of the prototype implementation, the emphasis turns to evaluating the implemented solutions. Finally, after the discussion of the results and evaluation, a conclusion is drawn based on the findings of this thesis.

1.4 Thesis Outline

Following the introduction, Chapter 2 introduces fundamental knowledge of the current context. Thereon, related literature and existing solutions or approaches connected to the topic of this thesis are discussed in Chapter 3. In Chapter 4, the main requirements, resulting design, the technical architecture of the prototypes, and solutions are presented. Chapter 5 covers and describes how the solution prototypes are implemented and explains the functioning behind the different components. Regarding the defined evaluation criteria in Chapter 4, Chapter 6 discusses and evaluates the outcome of the designed and implemented solutions. After all, Chapter 7 reviews the results and achievements of this thesis.

Chapter 2

Background

The background chapter covers essential information to understand the relevant topics adequately. It delineates the notion and implications of digital referral marketing as a form of modern referral marketing in the context of digital marketing. In addition, the terminology of referral programs and systems is outlined. Thereon, several modern-day examples of referral programs are portrayed to give an idea of their typical structure and design. Eventually, the fundamental technical concepts are discussed and introduced.

2.1 Digital Referral Marketing

In the context of this thesis, the terms digital referral marketing, modern referral marketing, or online referral marketing [60, 145] describe the same notion of referral marketing in the era of digital marketing. Traditionally, referral marketing has been mentioned in the same breath as WOM marketing since they are closely related to each other [21, 28]. Consequently, also modern WOM or Electronic Word of Mouth *(EWOM)* [26, 67] are still inherently connected to digital referral marketing. Frequently these terms are used interchangeably to describe the same concept or idea. Moreover, digital referral marketing is strongly interconnected to newly emerged digital marketing areas such as SMM, UGC, SEO, or affiliate marketing and many more [69, 86, 103]. Generally, referral marketing or customer referral marketing describes practices and initiatives where existing customers introduce or refer new customers to a product or service of a company or firm. Often the involved parties are rewarded for their efforts by referral rewards [19, 68, 149]

2.2 Referral Programs and Systems

Usually, digital referral marketing is applied and implemented in referral programs (sometimes also denoted as customer referral, digital referral, social referral, or online referral programs). Referral programs are the catalysts for referral marketing as they define the dimensions and requirements for the referral processes accomplished by the involved parties.

The design and operation of referral programs entail multiple aspects like organization, budget planning, identifying the right customers, or evaluation of the eventual success of a program [77] that can vary in different ways [19, 65]. As mentioned before, in the case of referral programs, these dimensions have and will increase to become more time- and resource-intensive with the advancements in digital marketing. Hence, modern digital referral programs are often referred to as whole referral systems.

In the context of this thesis, the term referral program comprises all aspects of the design decisions of a digital referral marketing program except the underlying technical solution and implementation. The whole system, consisting of the technical solution and implementation in combination with the referral program, is referred to as the referral system. This specific separation is not universally applied [21]. However, the breakdown is reasonable for this work as its primary focus is technical implementation.

2.2.1 Examples of Referral Programs

As referral programs and systems are becoming more prominent, there are incrementally more good examples of such [88]. Even if many of the components mentioned above of a referral program can vary, most programs are structured and designed similarly. To give an idea of the design and structure of modern-day referral programs, the following paragraphs review a few renowned examples.

2.2.1.1 Dropbox Referral Program

Dropbox [45] has one of the oldest digital referral programs often referenced when discussing digital referral programs [21, 72]. Due to its successful program, Dropbox saw immense growth in its user base in a short period of time [87]. The program started in 2008 and is still active 15 years later [46].

The straightforward design is one of the success factors for the Dropbox referral program [87]. The referring[1] and the referred [2] Dropbox user receive additional storage space as a reward for every successful referral process. Additionally, Dropbox provides separate conditions or programs for different users. Dropbox Plus accounts can complete more referrals and earn more rewards.

2.2.1.2 PayPal Referral Program

PayPal [100] is a service payment provider [101] that provides another often referenced [21] referral program. The program has, similar to the Dropbox example, a simple design and structure (*cf.* Figure 2.1) [79]. Upon a successful referral, both the referrer[3] and the referee[4] get a predefined amount of fiat money as rewards to their PayPal wallet.

[1]The *referring* user is the person who starts the referral process and refers another person
[2]The *referred* user is the person who has been referred by the person who started the referral process
[3]The *referring* person are from now on also described as *referrer*
[4]The *referred* person are from now on also described as *referee*

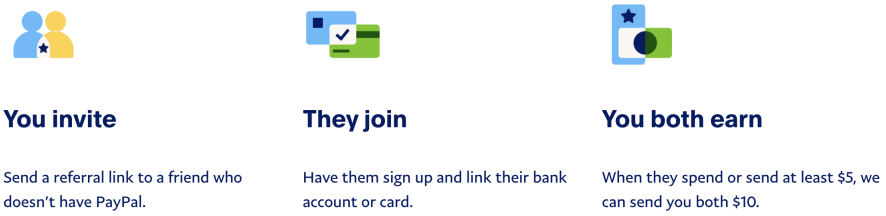


Figure 2.1: Screenshot of the PayPal Referral Process [99, 100]

2.2.1.3 Revolut Referral Programs

The list goes on with the referral programs of Revolut [117]. Revolut is a Financial Technology *(Fintech)* company that offers an alternative to traditional bank accounts in the form of a digital bank account on a mobile application [117]. First introduced in 2015, Revolut is operating in various countries and offering its services on a global scale [119]. Therefore, Revolut has introduced multiple referral programs over time, varying in their design and structure [82].

The goal and requirements for a successful referral or the rewards may differ depending on the country for which a program was designed. Another characteristic of Revolut referral programs is their temporal aspect, which forces users to complete the referral process within a certain period. For instance, one program would reward the users with fiat money, while for another program, a new or premium feature could be unlocked [82].

2.2.1.4 Coinbase Referral Program

The Coinbase [35] platform is a centralized exchange that allows the trading of cryptocurrencies [37] and offers a referral program to invite new users. Again, after the referred user has completed all the necessary tasks [36], the referrer and referee become eligible for the rewards. The Coinbase referral rewards consist of crypto assets that are directly transferred to the wallets of the involved parties. However, In contrast to *e.g.,* the example of Dropbox, Coinbase does exclude premium users or accounts from their referral program.

2.2.1.5 KuCoin Referral Program

KuCoin provides the last example of a referral program [74]. Like Coinbase, KuCoin is another centralized exchange that enables cryptocurrency trading. Yet, the KuCoin referral program differs from the Coinbase program. If referred users join the platform and complete tasks, the referrer is rewarded with stars as reward tokens or items (*cf.* Figure 2.2)[75]. Hence the more people a referrer invites and the more tasks the referees complete, the more stars can be earned. However, several limits and guidelines exist on how and when stars are distributed as rewards [76]. The earned stars can be used to buy prizes. Prizes can have different sizes and usually include some amount and type of crypto asset.

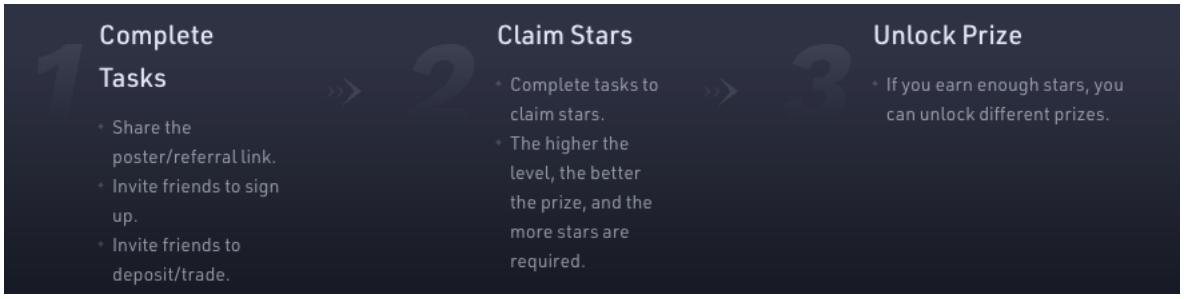


Figure 2.2: Screenshot of the Requirements of the KuCoin Referral Program [74, 75]

2.3 Technical Foundations and Concepts

This section delves into the technical concepts essential for understanding and implementing the eventual solutions. Thereon, the idea behind decentralized architectures and systems is presented in the context of this thesis. Eventually, related topics, including blockchain systems, Smart Contracts (*SCs*), as well as Decentralized Applications (*DApps*), and the notion of DeFi, are covered.

2.3.1 Decentralization vs Centralization

The procedure of decentralization [142] or centralization [140] is applied in various industries and is not exclusive to the field of technology. Nonetheless, the upcoming sections outline the implications of decentralization and centralization in the current context of software systems and applications [113]. In particular, this includes decentralized and centralized systems or architectures and solutions. In general, the architecture of a software system is either considered to be centralized, *e.g.,* with a traditional client-server structure [141], or decentralized, *e.g.,* in a Peer-To-Peer (*P2P*) based [144] system [44].

In some cases (*cf.* Drescher [44]), this differentiation is made between centralized and distributed systems. Overall, decentralized or distributed systems terminology is not fully consistent in the literature [137]. For one thing, the term distributed can describe aspects such as the computing power of a system, which can be geographically distributed among multiple machines. Thereon, centralized systems can be distributed as well [113]. Most modern centralized systems are distributed [40]. Conversely, the term distributed can refer to the control and decision-making power distributed over the system or network [44]. This case often describes the equivalent of a decentralized system. Sometimes, the talk is also about distributed decentralized systems [29]. However, this conception can be vague as well since often decentralized systems are, per se, seen as distributed. However, for this thesis, only the differentiation between centralized and decentralized systems is outlined and used from now on.

2.3.1.1 Centralized Systems

Essentially, centralized systems are characterized by a central entity or node that concentrates the authority over all the information, the data, the decision-making, and control at

a single midpoint of the system [143]. Other components of the system are dependent on this central entity. While centralized systems can make clear and fast decisions and have proven effective, they also provide a single point of failure and can be non-transparent [29, 140]. There exist various examples as the majority of modern software systems are centralized, *e.g.,* Google [59], Meta[5] [89], or Amazon [6], [113].

2.3.1.2 Decentralized Systems

Unlike centralized systems, decentralized systems have no single central authority [29, 44, 113]. Thus, information, data, and decision-making control are distributed to every participating entity rather than having a central node with all the power and control [143]. Among other advantages, decentralized systems can have a higher resilience or reliability and fault tolerance by not exposing a single point of failure [44]. Several areas and applications use decentralized systems. One example would be decentralized or distributed file-sharing systems or storage solutions like InterPlanetary File System (*IPFS*) [66], or BitTorrent [24]. Decentralized Autonomous Organizations (*DAOs*) [53] is another trending use case for decentralized architectures. Indeed, the most famous application of decentralized systems can be found in Distributed Ledger Technology (DLT), such as blockchains and blockchain-based cryptocurrencies.

2.3.1.3 Sybil Attacks

Sybil attacks were introduced by Douceur and refer to an attack or process where an attacker creates various fake identities to find or exploit vulnerabilities in a system [43]. Sybil attacks have been investigated as harmful in decentralized networks or systems [70] and on blockchain systems [23]. Moreover, Sybil attacks can be combined with other attacks or security threats [152]. Especially in decentralized systems and environments, a Sybil attack poses a potential security threat.

2.3.2 Blockchain Systems

Blockchain systems rely on the distributed ledger, or blockchain technology widely researched in literature [3, 42, 113]. A distributed ledger or blockchain is a decentralized, digital ledger of transactions that are trustfully and unalterably recorded, stored, and verified across a network with the help of consensus algorithms [90]. There are several attributes by which blockchain systems can be classified [44]. Depending on the reading and consensus access a system offers, four different forms of blockchains or distributed ledger systems can be determined. However, in most cases, the notion of a blockchain inherently refers to a public and permissionless distributed ledger. Blockchains provide traceability, immutability, distributed trust, security, and transparency in the form of a decentralized system [58].

[5]Better known under its former name: Facebook

Over time, blockchain systems have evolved and can be assigned into different eras [5, 124]. Initial solutions based on the first Bitcoin blockchain [92] have been followed by smart contracts capable blockchains such as Ethereum [27] or the Binance Smart Chain [22]. Additionally, modern solutions apply different or hybrid consensus algorithms to mitigate or eliminate known disadvantages of blockchains [58, 108].

2.3.3 Smart Contracts

Smart Contracts are automated chunks of code that can be immutably stored and executed on smart-contract-capable blockchains [54]. They provide a way to automatically transfer digital assets based on predefined conditions securely, traceably, and trusted between untrusted parties without needing a third-party intermediary [5, 27]. Not all blockchains support smart contracts to the same degree and in the same way. Several well-known blockchains that support smart contracts are Ethereum, the Binance Smart Chain, and Solana. If a blockchain supports smart contracts, it can automate rules and the distribution of assets in a decentralized way. Thereafter, smart contracts are the foundation for designing, developing, and implementing DApps or DAOs. Use cases for smart contracts can be found in several industries like healthcare, insurance, e-commerce, or the Internet Of Things (IOT) area [91]. Moreover, the most famous and widespread application of smart contracts until today is in the crypto sector *i.e.,* in DeFi apps or other cryptocurrency-based platforms.

2.3.4 Ethereum Virtual Machine

The Ethereum Virtual Machine *(EVM)* is the computer that is underlying the Ethereum blockchain and protocol, enabling smart contracts [85, 148]. It compiles and executes smart contract code [41]. The term gas is used as a unit to describe the computational effort the EVM requires to execute transactions. The EVM is not solely used on the Ethereum blockchain and protocol. Multiple other EVM-based or EVM-compatible blockchains utilize it, such as the Binance Smart Chain, Polygon, Arbitrum, Avalanche, and more. The EVM is included in the implementation or in layers that are added to make the blockchains compatible with the EVM. This enables these blockchains to support smart contracts and DApps similar to Ethereum.

2.3.5 EVM Values and Units

The native cryptocurrency used on Ethereum is Ether (ETH) and has several subdenominations [148]. Thereby, the smallest unit is called Wei. The following list shows the most common units used for Ethereum values.

- **Wei:** Wei is the smallest unit of Ether.
- **Gwei:** Gwei (Giga-Wei) is a larger unit of Ether, where 1 Gwei equals 10^9 Wei.

- **Ether:** Ether is the largest unit and is equal to 10^{18} Wei or 10^{9} Gwei.

Blockchains other than Ethereum that are EVM-based utilize the same unit system. However, sometimes the name of the main unit *i.e.,* , Ether in the case of Ethereum, is adapted. For example, in the EVM-based blockchain Polygon, the native currency is called MATIC [105]. Thus, one MATIC equals 10^{18} Wei on the Polygon chain.

2.3.6 Layer-2 Solutions

The term layer-2 solutions refers to blockchain solutions, protocols, or frameworks that have been designed to address the known problem of blockchain scalability by handling and executing transactions off-chain [56]. There exist multiple approaches and designs for layer-2 solutions [124]. A few examples of existing layer-2 blockchains are Polygon, Arbitrum, or Optimism. Moreover, layer-2 blockchains often have lower gas prices due to the reduced load, faster transactions, and overall improved efficiency [56, 124].

2.3.7 Blockchain Oracles

Blockchain oracles provide external off-chain data to blockchains and on-chain smart contracts [139]. Oracles are not the source of external data but rather the layer that collects, queries, verifies, or authenticates external off-chain data from various sources. This data is then fed to smart contracts on the blockchain. Blockchain oracles can be classified according to their data source, information direction, or degree of trust [20]. Inside blockchain-based systems and applications, oracles can represent potentially untrusted components as they might deliver inaccurate, corrupt, or malicious data from external sources that influence decisions on immutable smart contracts [30]. This predicament is generally known and discussed as the blockchain oracle problem [4, 30, 31]. Especially in the case of centralized blockchain oracle services, it is hard to ensure that the data and the oracle itself are trustworthy and impartial. Thus, approaches and solutions exist that aim to provide decentralized blockchain oracles [2]. Overall there exist various known approaches implementing centralized as well as decentralized solutions for blockchain oracles [2, 33, 102, 131]

2.3.8 Decentralized Finance

Decentralized finance, short DeFi, describes decentralized platforms and applications that provide blockchain- and smart contract-based financial infrastructure or services [122]. Within the crypto world, including digital assets and cryptocurrency markets, the DeFi sector has been growing significantly in recent years [111]. Empowered through decentralized smart contracts and blockchains, DeFi aims to supersede financial intermediaries as they are present in the traditional financial markets of today [112]. DeFi applications and platforms have various business models, including decentralized currencies, payment services, and fundraising [34]. Thereunder, Decentralized Exchanges (*DEX*) like Uniswap

[136], or Sushiswap [130] probably are the most known representatives for DeFi. A DEX allows users to trade, transfer and exchange digital assets.

Chapter 3

Related Work

The following sections explore existing literature and relevant work around referral marketing programs and systems. The main attention is on the design of referral programs and systems. More precisely, on the conceptual design of referral programs and the technical design or architecture of referral systems. Furthermore, it is discussed how referral programs and systems can be evaluated. Eventually, the focus shifts to existing solutions, approaches, and challenges arising in decentralized referral systems.

3.1 Design of Referral Programs and Systems

The following section is divided into two parts. First, the conceptual design of referral programs is covered and examined. The conceptual design contains all components of a referral program's design and structure but the technical aspects and architecture or details about their implementation. These are discussed in a separate paragraph dealing with the technical design of referral programs, *i.e.,* the design of the referral system.

3.1.1 Conceptual Design of Referral Programs

As suggested earlier (*cf.* Section 1.1), designing referral programs includes multiple components. There exists various literature investigating these aspects. Yet some aspects are more popular than others. To begin with, the work of Berman provides a comprehensive summary, including an eight-step overview (*cf.* Figure 3.1) that describes the topics which have to be considered when designing a referral program [21]. However, not all aspects of Berman are equally relevant for the case of this thesis.

3.1.1.1 Organization and Planning

The first two steps of the process (*cf.* Figure 3.1) describe the organization and the budget planning of a referral program [21]. The organization of a referral program inside

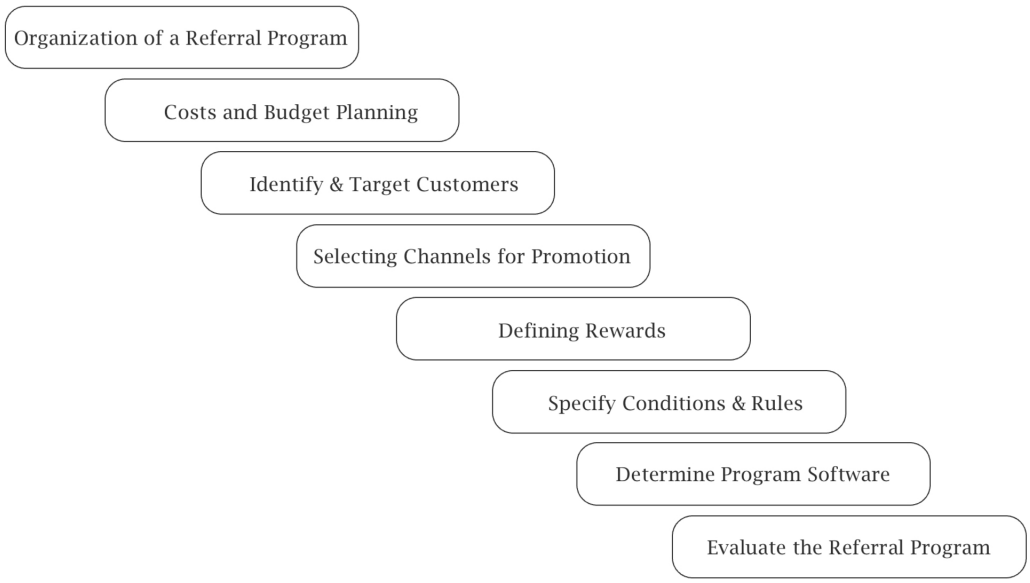


Figure 3.1: Own Illustration of the Eight-Step Process According to Berman [21]

a firm and the corresponding resources, as well as decisions about outsourcing, according to Berman [21], are not relevant in the current context. Moreover, this academic thesis has no genuine budget constraints regarding concrete planning costs. Nevertheless, reward costs for the conceptual design of the program or administration costs for the implemented software emerging from the chosen technical design must be considered.

3.1.1.2 Targeting and Selecting Referral Customers

When designing a referral program, selecting and targeting the right customers is vital. Apart from Berman, other research has covered this topic. Thereby, the value of customer referrals is often investigated [39, 63, 64]. Based on the work of Kumar et al. [78], Kumar, Petersen, and Leone calculate a Customer Referral Value *(CRV)* and argue how both Customer Lifetime Value *(CLV)*, as well as CRV, should be considered to identify lucrative customers for referral systems [77]. Furthermore, Schmitt, Skiera, and Bulte investigate factors like contribution margin, retention rate, and value of referred customers compared to non-referred customers [123]. Eventually, the customers or users of a referral program or system are a key element to consider when designing a referral program. The customers' expected motivation and attitude can mutually influence and be influenced by other components of a referral program's design, like the definition of rewards [25].

3.1.1.3 Referral Rewards

Referral Rewards have been thoroughly researched since they can significantly influence the outcome and design of referral programs. The allocation of rewards and its effects are explored together with the motivation and attitude of customers towards referrals [25, 68], pricing decisions and strategy of a product or service [25, 60, 149], or the conversion rates

and the total number of referrals [68]. Generally, there are three different approaches to the design of reward allocation:

- **Rewarding the Referrer**: Only the referrer receives the reward. The referee does not receive a reward.
- **Sharing the Reward**: The reward is shared between the referrer and the referee. There can be different distributions of the reward when sharing, but usually, the split is equal.
- **Rewarding the Referee**: Only the referee receives the reward. The referrer does not receive a reward.

On the one hand, rewarding the referee or equally shared rewards can lead to higher conversion rates, while on the other hand, prioritizing the referrer can cause more referrals but diminish the conversion rate as Jung et al. illustrate [68]. Universally, firms or companies should consider testing which design works best for their targeted customers as the allocation might be dynamically adjusted or set arbitrarily [21]. The allocation of rewards is relevant for this thesis and the eventual solution as it can have effects on the conceptual and especially technical design, *e.g.,* the distribution of rewards.

When and how to distribute rewards to the participants presents another design decision that can vary [19]. Thus, it is important to define and clarify the distribution schedule, *e.g.,* if rewards are paid out after every successful referral or only after a specific number of successful referrals.

Furthermore, the type and form of reward play an important role in the distribution of rewards. Referral rewards can come in the form of monetary incentives like fiat money and credits or crypto assets, vouchers or discounts, premium features or services, and many more [19, 21, 73]. The type and form consequently impact the distribution of the rewards to the eligible parties. For example, physical referral rewards like a vase or a teacup would be costlier to distribute than non-tangible bounties like additional storage space [46, 87]. Thereon, it is interesting to assess the examples of existing referral programs in section 2.2.1. These programs reward users with bounties or products based on their main business area. Dropbox offers additional storage [46]. PayPal, a payment service provider, and Revolut, a digital bank, offer fiat money as rewards [99, 116]. Moreover, the two centralized cryptocurrency exchanges, Coinbase and KuCoin, incentivize their users with crypto assets to complete their referral processes [36, 75]. Lastly, this also shows how the forms of rewards these companies have defined for their referral programs allow them to distribute the rewards efficiently to their customers.

Altogether, design decisions around referral rewards and their impacts should be well thought out and specified in the conditions or rules of a referral program to provide clarity for the participants.

3.1.1.4 Program Conditions and Requirements

The structure and rules of a referral program are defined by its conditions. A referral program's conditions should outline the concept and design decisions relevant to the participating customers. Thus, the conditions specify components like when a referral process is considered a success, who is eligible for referral rewards, how much rewards can be earned, how and when they are distributed, what actions are required by the participants or also general terms and restrictions [21]. Based on the program's conditions, participating customers complete several tasks and actions to fulfill all requirements to become eligible for rewards. On the one hand, these actions and requirements can be short-lived and simple, like in the aforementioned example of Dropbox, where referred users have to sign up for the platform (*cf.* Dropbox example section2.2.1.1). On the other hand, more steps, such as connecting a bank account or credit card or completing a certain amount of transactions and payments (*cf.* Section 2.2.1), can be required to receive rewards. Additionally, the conditions of a referral program must be well-designed and suitable for the company's use case.

Lacking conditions and rules for a referral program can open the door for malicious activities and referral fraud [9]. Not only in the current context but for modern referral programs in general are the conditions and rules of a modern referral program vital to the underlying technical implementation since they define the requirements for the evaluation of the referral process as well as the distribution of the rewards. In this context, this thesis explores the implications of decentralization on the design of referral program conditions (*cf.* Chapter 4). Thereon, the referral conditions play an essential role and must consider aspects and challenges that can arise in decentralized environments.

3.1.1.5 Referral Channels and Program Content

Berman [21] demonstrates three other design aspects of referral programs (*cf.* Figure 3.1). Decisions about the program software (*cf.* Section 3.1.2) or the evaluation of a referral program or system (*cf.* Section 3.2) are discussed in the upcoming sections. For this thesis, the definition of a program's content and promotion channels [21] is the last component that is illustrated as part of the conceptual design.

Promotional channels and contents shape a referral program's User Experience (UX). Simplicity and UX are key components for a successful referral program [79, 87]. Hence, the impact of referral channels and communication tools have been researched regarding referral propensity [1][72]. In fact, even minor elements like the naming of a program can impact the results of a referral program [138]. For this thesis, the focus lies on the technical architecture and implementation. Still, for future work, the UX and related aspects of the implementation can play an important role.

[1] How likely it is for a consumer to refer another consumer *cf.* [72]

3.1.2 Technical Design of Referral Systems

Regarding related literature that covers the design and technical architecture of referral programs, *i.e.,* the design of referral systems, there is little to no existing work. Yet, especially with modern and automated referral systems, the software and the technical architecture are important parts of a referral system that handle various tasks [21]. Generating referral codes, selecting and contacting the right customers, tracking and evaluating referrals, computing and distributing rewards, or evaluating the costs and results are a few examples of things that can be handled by the referral system. Nonetheless, not all of these aspects are relevant for a technical solution in the current decentralized context. Chapter 4 goes into more detail and elaborates on which functionalities are being implemented as part of the final solution.

Berman [21] lists a few examples of software packages for referrals like Ambassador [7] or ReferralCandy [114] and there exist more [81, 83]. However, their architecture and implementation are not open-source, as it is part of their business secret. The situation is similar with regard to the architecture and implementation of the examples mentioned in section 2.2.1. On this ground, it can be assumed that these referral systems are implemented based on centralized architectures. Thereon, all the logic, and functionality of the referral programs are handled by central servers [3]. Against this backdrop, this thesis feasibility study of a decentralized referral system might disclose interesting insights and arguments for or against this imbalanced current situation.

3.2 Evaluation of Referral Programs and Systems

Various metrics can be computed and collected to evaluate a referral program's success or ill success. Frequently, customer-related measures, such as the total number of participants or referrals, the CLV and CRV [63, 77], the conversion and retention rates [123], or acquisition costs [65] for referred customers compared to non-refereed customers are monitored. At the same time, revenues and expenses can be considered to determine the efficiency of a referral program [71] In addition, it is important to assess the positive effects of referral programs against the overall value or benefits that result for the company or firm behind the program [21].

However, no existing literature covers the evaluation of technical aspects of modern referral systems. As stated in the previous section (*cf.* Section 3.2), this is no surprise since commercial referral systems and their implementation are not open-source. Nonetheless, there are metrics and ways decentralized solutions can be evaluated in general regarding their costs and performance [124, 153]. More details about how the outcome of this work is evaluated and compared is discussed in Chapter 4 and 6

3.3 Classification of Referral Systems

Referral programs or systems can be classified in different ways [21]. The separation can be made with regard to the allocation of rewards *e.g.,* one-sided or equal split of rewards, the purpose behind a program *e.g.,* refer potential new employees or a product, or other design aspects that have been discussed in the previous sections (*cf.* Section 3.1.1 or 3.1.2). For this thesis, three different characteristics are considered for differentiating referral systems and their design. They are classified regarding qualified users *i.e.,* who can join a program and refer users via the system, allocation of the rewards, and in terms of reward levels *i.e.,* on how many levels rewards can be distributed.

3.3.1 Qualified Users

The notion of qualified users refers to the type and kind of users who are allowed or admitted to participate in a referral system. There can be limitations on the referring as well as the referred users. However, in most cases, restrictions on the people or users who can be referred would be counterproductive to the general goal of referral marketing.

- **Referring Users / Referrers:** Who and what kind of users are allowed to refer other users? *e.g.,* only a few selected or approved users, only current customers, or anyone, can participate *i.e.,* no limitations on the referring users.
- **Referred Users / Referees:** Who and what kind of users are allowed to be referred? Most of the time, anyone is allowed to participate as everyone could be a potential new user *i.e.,* no restrictions for referred users.

3.3.2 Allocation of Rewards

Referral systems can also be classified by how the rewards are allocated and distributed to the involved parties (*cf.* Section 3.1.1.3). For the current context, referral systems are classified as having either a one-sided or two-sided reward allocation.

- **One-Sided Reward Allocation:** Either the referrer or the Referee receives rewards, but not both.
- **Two-Sided Reward Allocation:** Both parties receive rewards. Thereby the ratio of the split can vary.

3.3.3 Reward Levels

The third criteria to classify referral systems in this context are reward levels. Depending on the reward level of a referral system *i.e.,* on how many levels the rewards are distributed, a system is either classified as single-level or as multilevel.

The main characteristic of multilevel referral systems is that rewards can be distributed on more than one level. Referred users can refer new users, who in turn can again refer users, and so on. Consequently, a referral chain is formed containing and storing the different levels or tiers of referrals. Referral rewards can then be distributed over this chain and along multiple levels to previous referrers. However, most referral systems include only one level of rewards. Hence, only the two involved parties on the first level are rewarded *i.e.,* the referrer and or the referee.

- **Single-Level Referral System:** Rewards are assigned and distributed on one level only. For example, only to the referrer and the referee.
- **Multilevel Referral Systems:** Rewards are assigned and distributed on more than one level. Hence, previous referrers along the referral chain also receive rewards when referred users refer new users.

This classification aims to help differentiate and arrange existing referral programs or systems as well as the upcoming solutions in the context of this thesis.

3.4 Decentralized Referral Systems

Until this point, academic literature has not covered the possibility and design of decentralized referral systems or their evaluation. However, there exist projects that have implemented decentralized referral programs or related solutions. This section introduces three of these examples and their approaches and design. In addition, this section addresses research that provides insights into how to realize a decentralized referral system. To be more precise, it contextualizes the blockchain scalability trilemma into the current problem statement of decentralized referral systems dealing with potentially high volumes of data and transactions.

3.4.1 Existing Decentralized Solutions

Besides academic literature and work, a few projects are trying to provide ways for decentralized referral marketing. Three of these examples, namely Attrace [18], Energi [50], and RefToken [115] are outlined in the following paragraphs. Other than these, there are potentially more projects around this topic that can be found. However, many of these projects are only in the testing stages, outdated, or do not have proper documentation where more details about their functioning can be found.

3.4.1.1 Attrace Referral Protocol

Attrace [18] provides a referral protocol targeted for Web3 [146], where users can share referral links to earn rewards and promote existing crypto projects. The Attrace protocol

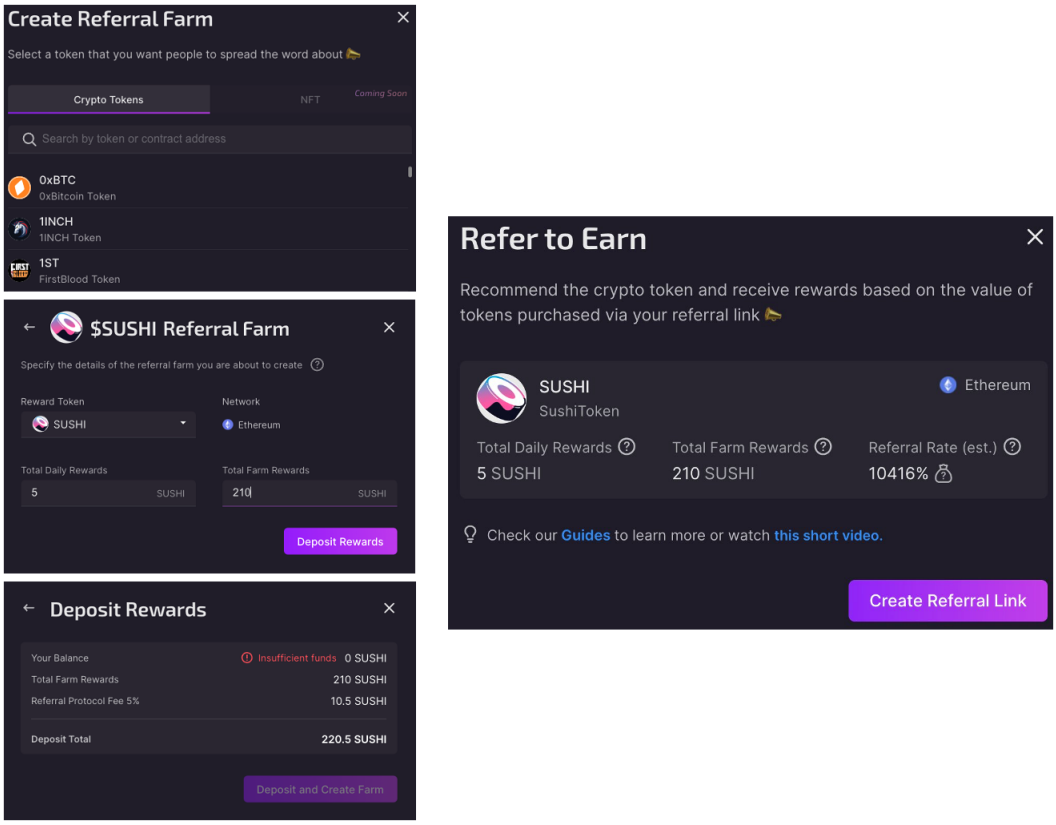


Figure 3.2: Own Illustration of the Farm Creation Process by Attrace [18]

powers a decentralized referral platform based on smart contracts and blockchain oracles [12]. Attrace aims to help increase transparency and trust and allows for lower fees and faster payments than traditional marketing.

Attrace Referral Farms: On the Attrace web platform, users *i.e.,* project owners, can create so-called referral farms [17], in which they have to define a crypto project or Non-Fungible-Token (*NFT*) that they want to refer. For this farm, they have to define several conditions, like the type of crypto asset for the reward, the number of daily rewards available, and a marketplace where the promoted crypto asset can be traded. Thereafter, the project owner has to deposit the rewards to the farm and receives a referral link that can be shared with others. The Attrace referral farm creation process is illustrated in Figure 3.2. The setup of referral farms is handled decentrally by the *ReferalFarms* smart contract, which Attrace has deployed to the Ethereum blockchain.

Promoters / Referrers: Next, Attrace users or referrers can select crypto projects and farms they want to promote. To become a promoter of a farm *i.e.,* a crypto project, they have to sign in to Attrace with their crypto wallet. Thereon, they receive a referral link that can be shared with potential buyers and other users.

Buyers / Referees: Users who have been referred to a crypto project are called buyers by Attrace. A referral process is completed once buyers have followed the referral link, signed in with their wallet, and bought the promoted crypto asset on the marketplace, which has been specified in the setup of the referral form.

Oracles / Referral Evaluation: After the completion, the referral process and the value added to the promoted crypto project are periodically evaluated by the Attrace oracles [16]. Attrace oracles are organized as an off-chain network that does not require paying gas fees for made transactions. The main function of the oracle nodes is to observe, collect, evaluate, and calculate relevant data about the referral processes and the spread of rewards, which is then delivered to the *ReferalFarms* contract (*cf.* Section 2.3.7). Hence, the oracles evaluate the referral process and the calculated rewards to the responsible, smart contract. Eventually, Attrace oracle nodes are envisioned to be part of a public and decentralized oracle network that everyone can join after staking the required amount of $ATTR tokens [15]. The $ATTR token is a native utility and governance token by Attrace [14]. However, according to their roadmap [13], the oracle network is not yet public. Therefore, the evaluation of the referral processes is also not open-source. Hence, as of today, the Attrace solution is not completely decentralized.

Rewards: The calculated Attrace referral rewards are assigned to the promoter as well as to the buyer with regard to the conditions and proportions of the referral farm. Rewards are distributed by the *ReferalFarms* contract based on the results that have been defined with the help of the oracles.

In conclusion, Attrace aims to provide a decentralized referral program. Users can buy crypto tokens or NFTs via a predefined marketplace. The *ReferalFarms* smart contract, together with the first versions of referral farms and deals, are already available and ready to use on their platform [18]. Still, this is only the first of a dozen milestones that Attrace has defined in their roadmap to provide a fully decentralized referral protocol [13]. The Attrace referral system has no restrictions concerning qualified users, the reward allocation is two-sided, and the system distributes rewards over one level (*cf.* Section 3.3). Attrace plans to achieve decentralization through smart contracts on the Ethereum chain and a public oracle network. Nevertheless, even if referral and reward data may be stored decentrally, with the evaluation of the referral processes and the calculation of rewards, major parts of their system are not yet decentralized and are also not open-source.

3.4.1.2 Energiswap Affiliate Program

Energiswap [49] is a DEX based on the Energi blockchain that serves as the base for the Energi World ecosystem [51]. The Energi ecosystem offers its own blockchain oriented and compatible with Ethereum, including EVM compatibility, virtually negligible fees, and other features [48]. Thus, smart contracts written for Ethereum also work when deployed on the Energi blockchain. In addition, Energi provides a block explorer to observe Energi transactions and a bridge that allows transferring assets between the Ethereum and Energi blockchains. Recently, Energi launched a decentralized referral or affiliate program within their Energiswap DEX [50].

Affiliate / Referral Conditions: Users can participate in the program via an affiliate page on the Energiswap web platform [49], where they can get a referral link to share with others (*cf.* Top in Figure 3.3). Referred users can follow the link and are directed to the same affiliate page again, but this time with the pre-filled address of the referring user (*cf.* Bottom in Figure 3.3). To accept an affiliate link, referees or referred users have to

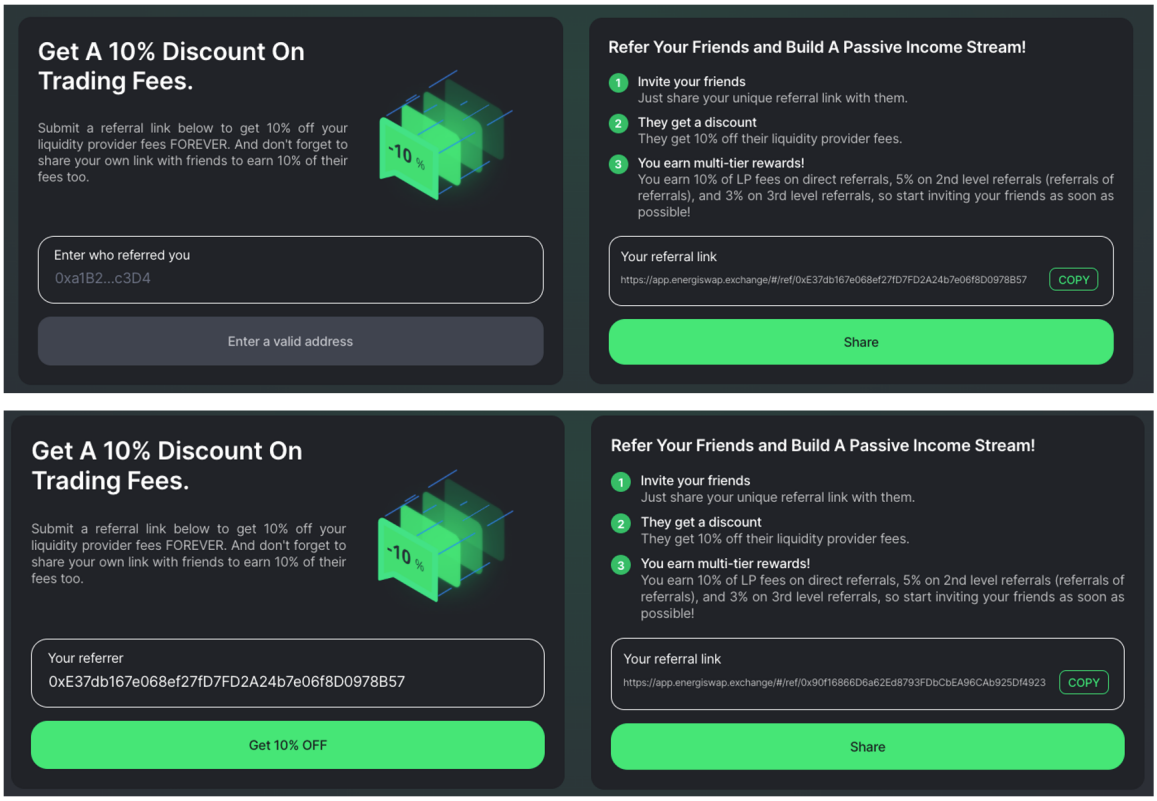


Figure 3.3: Screenshots of the Energiswap Affiliate Platform [49]

execute a `Set Referral (Address)` transaction to enter the referral process and redeem their potential discounts for future trades (*cf.* Left in Figure 3.4). The transaction costs gas fees (*cf.* Right in Figure 3.4) and stores the resulting referral connection on the Energi blockchain. Thus, acceptance of affiliate invitations via the links is decentrally tracked and stored.

Thereon referred users permanently receive a 10% discount on the trading fees for their Liquidity Provider (*LP*) transactions on the Energiswap DEX. Conversely, the referrer benefits by receiving a 10% commission on trading fees paid and caused by LP transactions that their referred users have made on the DEX (*cf.* Figure 3.5 A). In this case, these discounts and commissions on made transactions are the rewards of the Energiswap affiliate program. Energiswap discounts and commissions are automatically calculated and applied or paid out to the eligible users by their referral system in a decentralized way [50]. In addition, the commissions are paid in the cryptocurrency used at every transaction's start. For instance, if a referred user trades cryptocurrency (A) for a cryptocurrency (B) on the Energiswap DEX, the commissions are paid out as assets of cryptocurrency (A).

Multi-Tier Rewards: The Energiswap affiliate or referral program provides multi-tier commissions [50]. Hence referrals can be made over multiple levels. Thereon, if users referred in the first place refer more users who join the program, the second tier receives a 5% discount or commission (*cf.* Figure 3.5 B). The same concept applies to the third tier of referrals, where the percentage for commissions and discounts is 3% (*cf.* Figure 3.5 C). Eventually, this builds a referral chain that further rewards users.

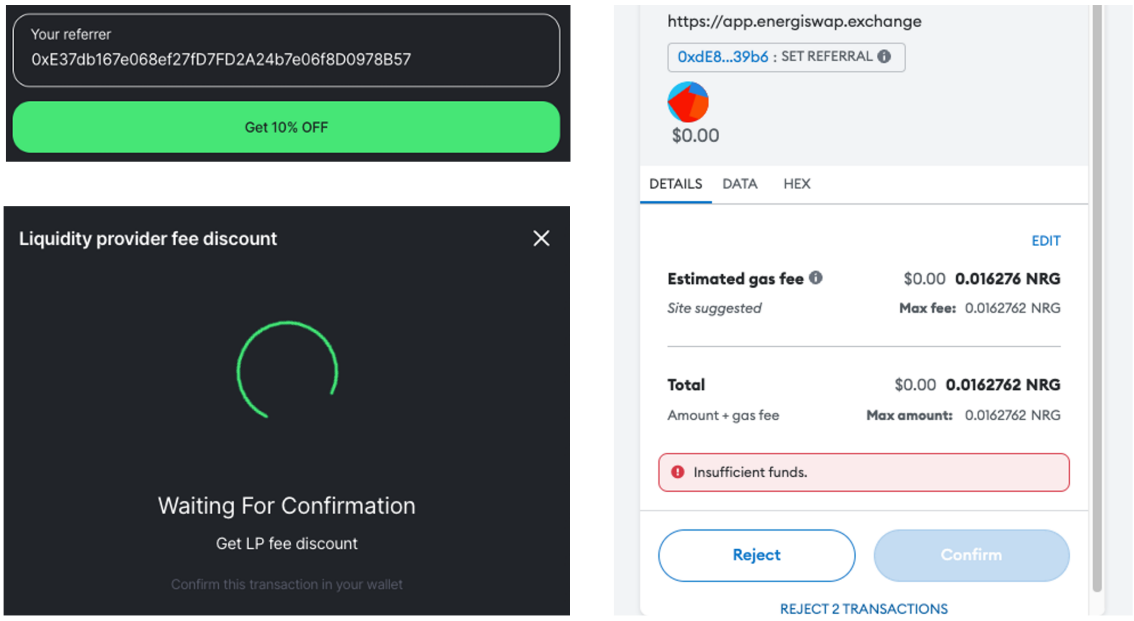


Figure 3.4: Screenshots of the Energiswap Referral Acceptance Transaction [49]

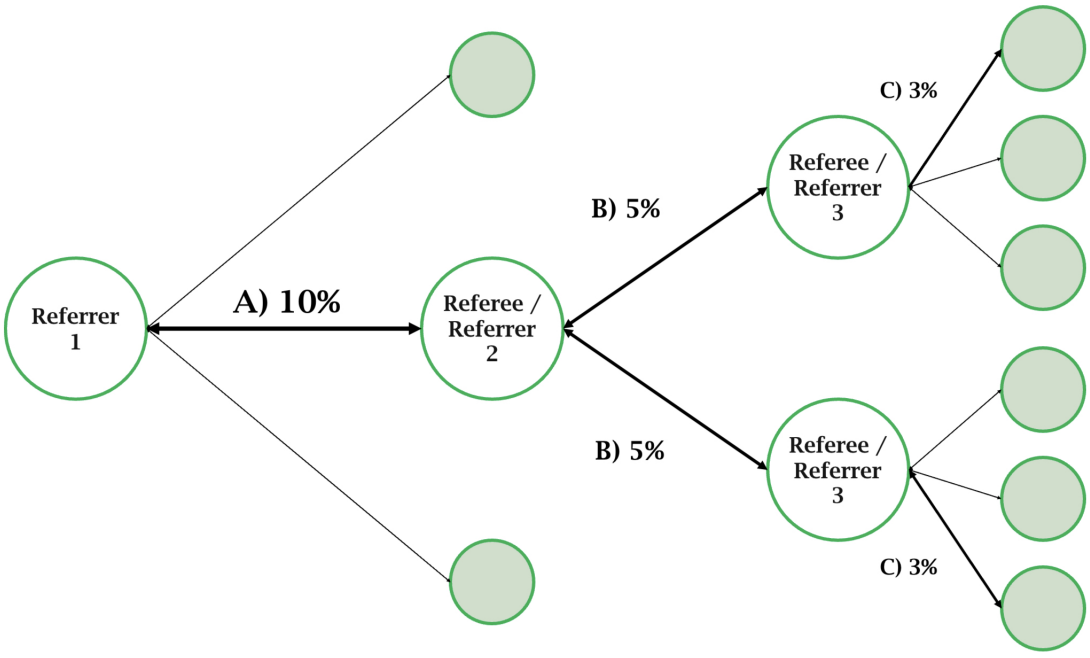


Figure 3.5: Own Illustration of the Multi-Tier Energi Referral Flow [50]

Eventually, the Energiswap program provides a multilevel referral system with two-sided rewards and allows everyone to participate (*cf.* Section 3.3). In contrast to Attrace [18], which works with different exchanges *e.g.,* Uniswap or Sushiswap, the Energi affiliate program is incorporated into the Energiswap DEX [50]. The referral system stores every user's referral chain on their Energi blockchain. Additionally, they have deployed smart contracts for reward allocation and distribution. Yet, in-depth documentation about these smart contracts, including more details about the decentralized evaluation of referral processes or the calculation and distribution of rewards, seems not to be available at first sight. This could be done on purpose since, except for their affiliate program, Energi [51] provides various open-source documentation about other components of their ecosystem [47]. Moreover, further research and exploration of the Energi blockchain might reveal more information about the deployed smart contracts and their details.

3.4.1.3 RefToken

RefToken [115] is or was a solution built to provide a decentralized referral platform. As of today, their platform is no longer available due to several reasons [10]. Still, RefToken has published a whitepaper [120] and a yellow paper [84] describing their initial business plan and technical details and architecture.

The RefToken solution is a blockchain-based affiliate marketing platform with the purpose of promoting mainly DApps and Initial Coin Offerings (*ICOs*) [120]. To achieve this, governance and evaluation of the referral processes are handled by smart contracts deployed to the Ethereum chain by RefToken. All referral-relevant data are immutably stored on the blockchain to allow tracking and prevention of referral fraud. Additionally, their smart contracts enable automatic and instant rewards payout to all eligible parties.

The following sections describe the different parties involved in the RefToken referral process and how the referral flow is outlined by RefToken in their yellow paper [84]. Moreover, the different smart contracts are described. Generally, three different parties are involved in the referral process according to RefToken (*cf.* Figure 3.6 A, B, and D).

Merchants / Program Providers: On the RefToken platform, merchants *i.e.,* companies or firms, can register and create deals that are then open for affiliates *e.g.,* referrers, to apply for (*cf.* Figure 3.6 A). A merchant admin website allows merchants to register and log in to create these deals. Besides, the merchants have more features and customization options available on their admin site [84].

For every published deal on their platform, RefToken deploys a *deal-affiliate link* smart contract containing details and conditions for the referrals. These contracts deployed to Ethereum include details about the deals, like the total amount of available rewards or deal deadlines. Referral rewards are always defined and eventually paid out as REF tokens, a cryptocurrency issued by RefToken.

The *deal-affiliate link* smart contract stores all the information about the deals and the approved affiliates, as the name implies. The contract also includes functions to update the authorized affiliates, edit deal details, or add custom conditions for certain affiliates.

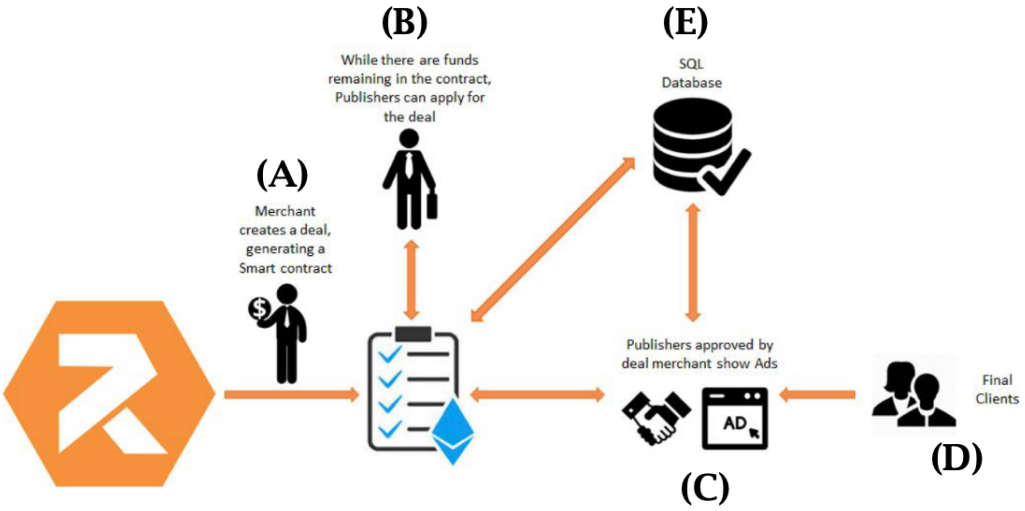


Figure 3.6: Extended Illustration of the RefToken Platform Flow According to RefToken [84, 115]

Affiliates / Referrers: Users who want to promote and refer the merchant's DApps or ICOs could then apply to become affiliates of the respective deals of a merchant [84]. Like the merchants, affiliates have their own admin platform where they can register, log in, and apply for deals. Thereby, affiliate users also have the possibility to apply for custom deals and negotiate the terms and conditions for referrals directly with the merchants. Every affiliate or referrer has to be accepted or approved by the deal's merchant. Furthermore, as long as the *deal-affiliate link* contract of a specific deal would still have funds left for referrals, new users can apply to become eligible to promote the deal (*cf.* Figure 3.6 B). After the approval, affiliates can promote the deals in order to reach the final clients by themselves or via other channels like ads or social media (*cf.* Figure 3.6 C).

Final Clients / Referees: Referred by the affiliates, the final clients *i.e.,* referees, can sign-up, buy something or in any other way make use of the merchant's DApps or participate in their ICO (*cf.* Figure 3.6 D). Eventually, as the final clients interact or engage with the merchant's DApps or buy coins in an ICO, the affiliates are automatically rewarded by the deployed *deal-affiliate link* contract. According to the conditions stored in the contract, the affiliates would receive different amounts of REF tokens. Besides, RefToken would also try to track the interaction of final clients coming from affiliates in their off-chain database (*cf.* Figure 3.6 D).

RefToken Smart Contracts: RefToken describes four different kinds of contracts that have to be deployed for their platform [84]. The first two contracts are the *token definition* and the *token exchange* contracts, which are needed for the issuance and operation of the REF token. The third *deal-affiliate link* contract has been discussed before. Last, the fourth type of contract considers the conditions of the deal.

RefToken describes the *deal-conditions* contracts as a set of multiple contracts with the purpose of covering and evaluating all the referral conditions of the different deals available on the RefToken platform. Hence, deal (A) might need a different *deal-conditions* contract than deal (B) since the referral conditions and evaluation between these two

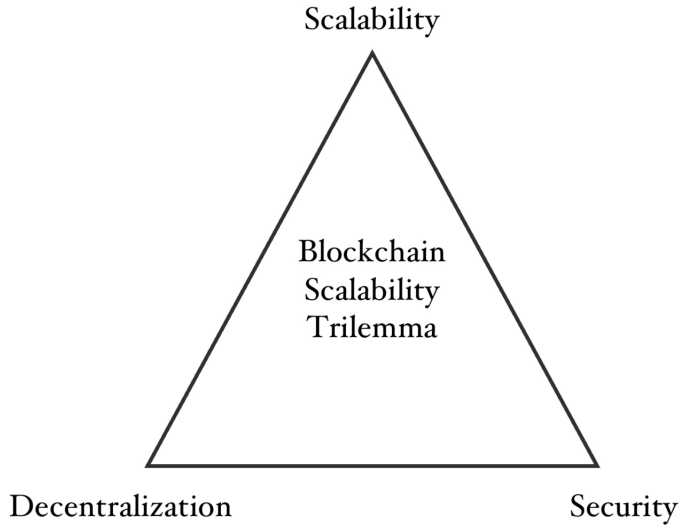


Figure 3.7: Own Illustration of the Blockchain Scalability Trilemma According to Sguanci, Spatafora, and Vergani [124]

deals differ. For example, a *deal-conditions* contract could require functionality to count the number of deposits to a particular address. RefToken mentions one example of an ICO contribution *deal-conditions* contract and outlines a potential deal flow for this case in their yellow paper [84]. Other than that, there is no more documentation about how the *deal-conditions* contract could enable the evaluation of the different referral processes or deals in a decentralized way.

After all, RefToken includes and outlines several interesting aspects of how a decentralized referral program could be realized. In their white [120] and yellow paper [84], they explain several aspects of their solution. Nevertheless, the platform can not be tested and tried out as the project is no longer running, and the different platforms are not available [10]. Moreover, the project seemed to be only in its early stages, which can also be seen in the available documentation.

3.4.2 The Blockchain Trilemma

This section covers the last important and related aspect discussed in this chapter. The blockchain or the blockchain scalability trilemma refers to the problematic limitation or trade-off for blockchain systems with regard to their scalability, security, and decentralization [150]. Only two of these three aspects can be reached by blockchain systems (*cf.* Figure 3.7). As the scalability of blockchain systems has been identified as the major problem, this topic has been addressed in various research [108, 153]. In order to address the limitations of blockchain scalability, novel approaches, and solutions have been and are still being researched in connection with blockchain technology [124]. Still, the trade-off between the desired properties has to be made as no silver bullet for solving the trilemma has been found so far.

In the context of this thesis, the feasibility of a decentralized high-volume referral system is investigated. Hence the degree of decentralization and scalability are important for the final solution. However, the trilemma already shows at this point. The solution must be secure since only a secure solution can underpin the eventual feasibility of a decentralized referral system in a real-world environment.

In summary, this chapter delves into various aspects of designing, evaluating, and implementing referral programs and systems. In terms of the design, existing literature covering conceptual design aspects has been outlined (*cf.* Section 3.1.1). Moreover, the differentiation that was made between referral programs and referral systems, or conceptual and technical design respectively, (*cf.* Section 2.2) shows that there is little to no research when it comes to the technical design of referral systems (*cf.* Section 3.1.2). Based on the previous two chapters, the scarcity of availability or documentation about non-centralized referral programs or systems is illustrated broadly. This gap becomes more prominent as there is also a notable absence of academic literature and technical documentation in the context of especially decentralized referral systems.

However, this chapter has introduced projects that try to provide solutions for decentralized referral systems (*cf.* Section 3.4). At this point, it is to mention that there might be other approaches or related solutions for decentralized referral systems, which have not been considered within the scope of this thesis. The three covered projects, namely Attrace [18], Energi [51], and RefToken [115], show various similarities but also differences in their approaches to solving the challenge of decentralized referral systems. All the projects have been started recently and are in their early stages. The available documentation for these projects provides insights into a few concepts that can be used for the design of decentralized referral systems. Nevertheless, these projects' overall documentation, design, and implementation remain lacking and sometimes non-transparent. Still, the general interest and desire for solutions for decentralized referral systems show within these projects.

With that in mind, this thesis aims to fill the gap and provide a contribution to academic work about decentralized referral systems' feasibility and provide transparent and extensive documentation about their technical design and implementation as well as challenges and implications for a potential application in real-world conditions.

Chapter 4

Design and Architecture

The design chapter covers and outlines the main requirements and the design of the final solution architecture for this thesis. In the first step, the primary requirements for decentralized referral systems are elaborated on and discussed within the scope of a decentralized environment. Thereon, the architecture, and design for eventual solutions are introduced and presented. After depicting the solution prototypes against the defined requirements, metrics and measures relevant to their evaluation are described.

4.1 Requirements

As discussed in the last chapter (*cf.* Chapter 3), there is little to no documentation available about the technical requirements of decentralized referral systems. Therefore, this thesis defines two fundamental requirements for referral systems or decentralized referral systems that must be fulfilled by any technical solution to be considered a referral system. Besides other important aspects that can be handled by a referral system (Section 3.1.2), these two specifications have to be met by the final solution in the context of this thesis:

- **Referral Process Evaluation:** The system must be able to recognize, track and evaluate multiple referral processes, including two or more users. Consequently, the system must be aware of the progress and state of every started referral process and determine which processes have been completed either successfully or unsuccessfully. Moreover, the system must be able to define the parties *i.e.,* users who have become eligible for potential referral rewards. Furthermore, the type and amount of reward to be distributed must be determined.

- **Distribution of Rewards:** The potential rewards of a referral process defined by the system and referral process evaluation in the first place must be distributed correctly to all eligible parties.

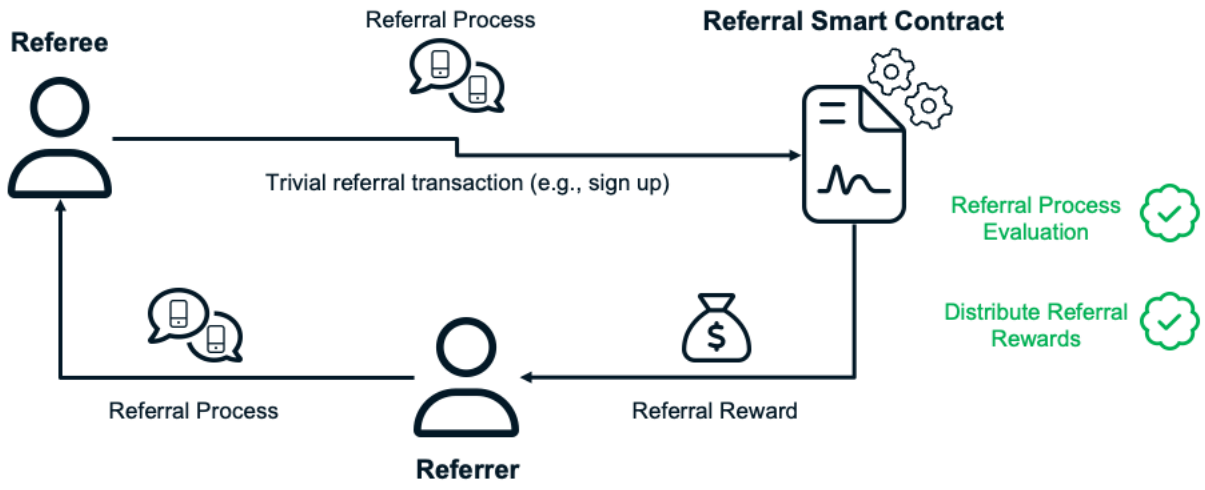


Figure 4.1: Design of a Trivial Decentralized Referral System

Based on these two requirements, designing a trivial but legitimate solution for a decentralized referral system is already possible. Figure 3.2 outlines a straightforward decentralized referral system oriented at the previously introduced referral program of Dropbox (*cf.* Section 2.2.1.1).

In this referral system, a decentrally deployed referral smart contract would handle the evaluation of referral processes and the distribution of rewards. Participants *i.e.,* referees, could send a transaction to the smart contract with information about another user who has referred the participant *i.e.,* the referrer. To keep it very simple, referees would complete the referral process after sending that first transaction with the required information about the referrer to the smart contract. Hence, the referral process or the referral conditions would imitate a registration process similar to the requirements of the Dropbox referral program (*cf.* Section 2.2.1.1). The process or registration is evaluated and marked as completed as the referral smart contract records the transaction. After the completion, a predefined reward could be distributed or sent by the smart contract via the blockchain to the referrer. The referee transaction *i.e.,* registration would have been evaluated, and the referrer would have received a referral reward based on the decentralized referral smart contract. Thus, the defined requirements would have been met (*cf.* Figure 4.1).

However, this example shines a light on one of the most crucial challenges or requirements for the eventual solution of this thesis. Since a decentralized referral system naturally operates in a decentralized environment, the term of a user of such a system is not synonymous with a user in a centralized referral system. A user usually corresponds to a single person in a centralized referral system. Additionally, in centralized environments, multiple ways exist to ensure or verify a one-to-one relationship between a referral system participant and a real person. However, in this example of a decentralized referral system, users refer to accounts or addresses on a blockchain (*cf.* Figure 4.2). Hence, a single person or wallet holder *i.e.,* person can have multiple accounts or addresses. Consequently, a single person could participate as referee and referrer simultaneously and drain all the rewards of the referral smart contract. As a result, a trivial Dropbox-like decentralized

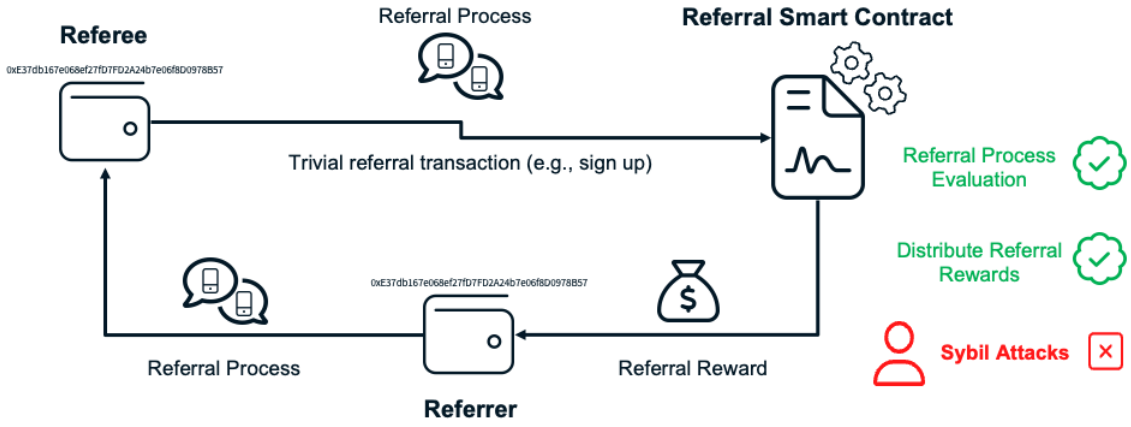


Figure 4.2: Decentralized Users *i.e.,* Addresses in a Decentralized Referral System

referral system similar to the one outlined in Figure 4.2 would be very easy to exploit and vulnerable to Sybil attacks (*cf.* Section 2.3.1.3) for example.

Ultimately, the system providers *i.e.,* smart contract owners, are not directly endangered by such Sybil attacks, as the referral rewards would consistently be distributed across multiple addresses in every scenario. Nevertheless, such a referral system would miss its primary marketing purpose and objective if the program and rewards solely benefit one malicious individual. Eventually, the architecture and design depicted in Figure 4.2 represent a logical but ultimately infeasible solution.

4.2 Solution Requirements

The previous section has exemplified a vital problem or requirement of decentralized referral systems. In a decentralized environment, additional security risks can invalidate the design of well-known referral systems or programs. Consequently, the design and referral conditions must be adapted to resist threats and attacks in decentralized environments. In this context, the final solution requirements for a decentralized referral system have to be adapted:

- **Referral Process Evaluation:** The system must be able to recognize, track and evaluate multiple referral processes, including two or more users. Consequently, the system must be aware of the progress and state of every started referral process and determine which processes have been completed either successfully or unsuccessfully. Moreover, the system must be able to, on the one hand, define the parties *i.e.,* users who have become eligible for potential referral rewards. On the other hand, determine the type and amount of reward to be distributed.
- **Distribution of Rewards:** The potential rewards of a referral process defined by the system and referral process evaluation in the first place must be distributed correctly to all eligible parties.

- **Adapted and Secure Design for Decentralized Environments:** The solution or system must be secure and impervious to decentralized security threats like Sybil attacks or other threats.

Besides, other requirements can be important for the success and quality of a referral system (*cf.* Section 3.1). Many essential qualities, like usability, accessibility, transparency of the referral conditions, clear feedback during the process, trust, and others, are related to the UX of referral systems. If applicable, these requirements also should be considered for the final solution prototypes and their implementation. Certainly, the main requirements covering more the technical aspect and architecture are prioritized.

4.3 Referral Payments

The architecture illustrated in Section 4.1 (*cf.* Figure 4.2) meets two of the three solution requirements. The remaining challenge to be tackled is enhancing resistance against malicious users that can invalidate the purpose and effect of such a referral system or program. Essentially, there are two strategies for addressing this issue.

The first approach is restricting access to the referral system by regulating users and ensuring each real-world person can only participate once, using a single address or any other form of identification in the program. While this can be relatively simple in centralized environments, it becomes a much more complex task in decentralized systems. Although there are methods to include this in decentralized systems *e.g.,* using blockchain oracles and off-chain data (*cf.* Section 2.3.7 or 3.4), these solutions can introduce centralized elements into the system, compromising the overall degree of decentralization.

The second approach is comparably more straightforward. Malicious actors exploit a system's vulnerabilities only when they can derive some form of usually financial benefit. Hence, adapting the design and architecture to make exploiting the system unprofitable or unattractive for users can prevent attacks similar to the previously discussed access control or restriction. Financial disincentives are a common approach to ensure security by design in decentralized environments. Moreover, this method does not result in any compromises in the system's degree of decentralization. As the ultimate goal is to create a solution that is as decentralized as possible, this second approach appears to be more suitable in the context of this thesis.

The proposed design solution involves allowing only referral payments or payment transactions within the referral process. The concept of referral payment transactions is depicted in Figure 4.3. Essentially, every referral transaction must include some form of financial assets to ensure the negative stimulus of these transactions for malicious users. There are no effects on the referral process related to how referees and referrers communicate or refer each other (*cf.* Figure 4.3 A).

Thereon, instead of arbitrary transactions *e.g.,* register transactions as illustrated in Figure 4.2, each transaction represents a payment and thus includes assets transferred from

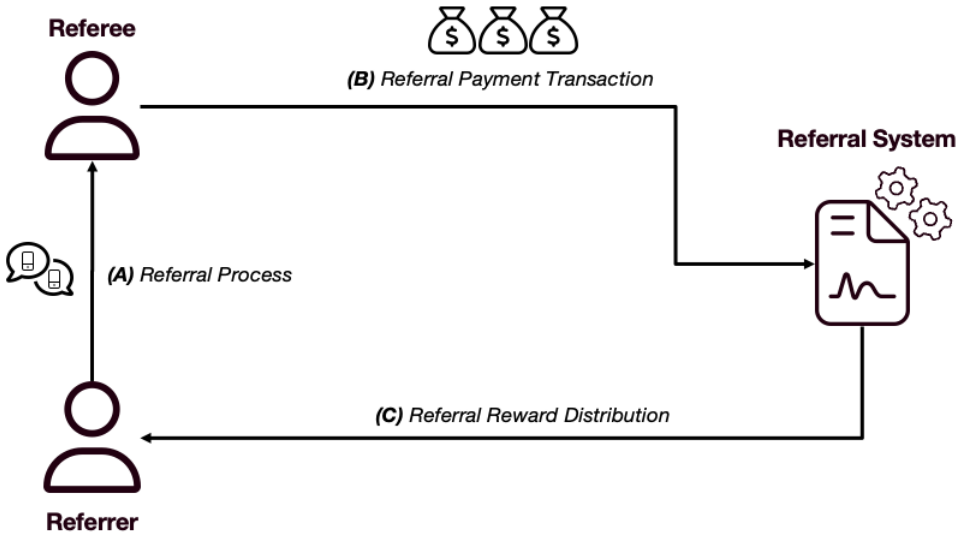


Figure 4.3: Conceptual Design of Referral Payment Transactions

the participating referee to another address (*cf.* Figure 4.3 B). Consequently, participation in the referral system and sending transactions comes with costs. Such payment transactions have been, and continue to be, a common measure or requirement in various referral program conditions (*cf.* Section 2.2.1.3). In this way, the referral process is connected to user payments when purchasing any service or product. Legitimate users can benefit from the referral program and potential rewards as cash back for purchases (*cf.* Figure 4.3 C). Malicious users, conversely, are unlikely to exploit the system since every referral payment transaction is associated with costs. Ultimately, it is possible to implement a design where the rewards do not outweigh the costs of the payment and therefore discourage potential Sybil attacks.

4.4 Deferral - Solution Architecture and Design

This section outlines Deferral, including its fundamental architecture for decentralized referral systems and the design of the different solution contracts. The Deferral solution relates not only to one design but to a set of potential solution designs for smart contracts that share a common architecture. They differ in a few aspects related to their design, especially from the technical implementation point of view. However, a common base architecture is shared among all the solution designs. A more technical deep-dive into the various prototype designs and implementations can be found in Chapter 5.

4.4.1 Referral Payment Evaluator

The final solution architecture is displayed in Figure 4.4 and extends to the concept of referral payments (*cf.* Figure 4.3). The fundamental solution architecture remains straightforward. At the core of the architecture is the referral payment evaluator component.

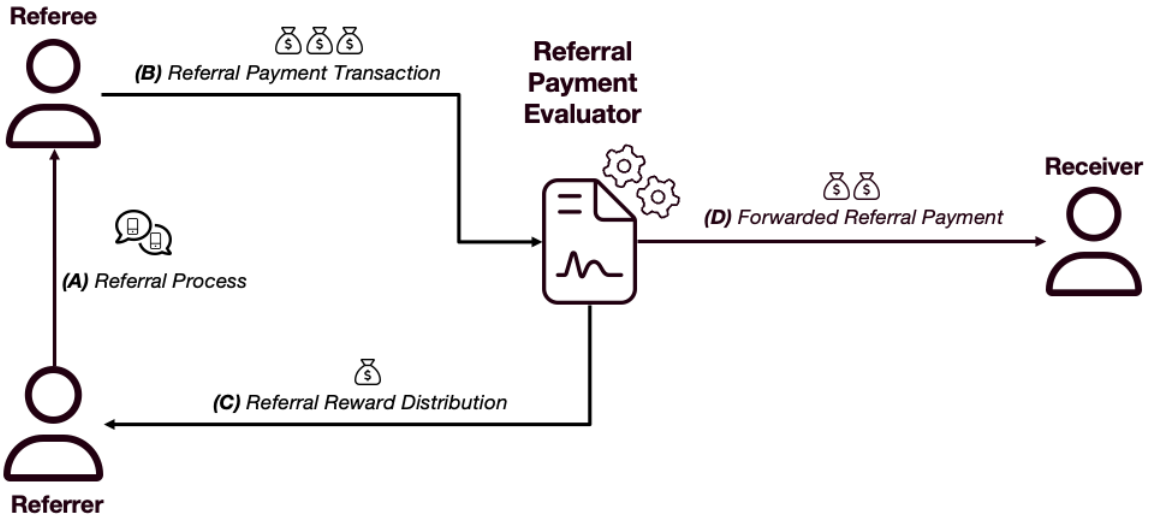


Figure 4.4: Deferral Solution Architecture

The referral payment evaluator is a smart contract that can be deployed decentrally on a blockchain that supports smart contracts (*cf.* Chapter 2). The evaluator smart contract fulfills multiple functions.

Firstly, it acts as a kind of proxy or transmitter for the payment transactions received by the referees (*cf.* Figure 4.3 B). Referral payment transactions sent to the contract are accepted or declined and subsequently forwarded to an address, *i.e.,* a user, stored on the contract (*cf.* Figure 4.3 C). This receiver represents the wallet or account of the company responsible for the referral system, which would deploy and operate it. The referral payment evaluator contract is designed to implement the referral payment concept discussed before. A referral system with this chosen design can be implemented in any use case or scenario where a company accepts cryptocurrency payments.

Second, the evaluator smart contract is responsible for evaluating the referral process. Based on the recorded payment transactions, it collects the data necessary to evaluate the referral processes against the defined referral conditions. The design of the referral conditions is discussed in more detail in the upcoming sections. Lastly, the referral payment evaluator contract is also responsible for distributing the rewards. The users or addresses eligible for rewards and the number of rewards are determined by the process evaluation and eventually distributed by the smart contract.

Besides the referral payment evaluator component *i.e.,* smart contract, other entities exist in the solution architecture *e.g.,* the receiver, referrer, or referee. However, most solution designs introduced in the next section mainly affect the referral payment evaluator contract and its implementation.

4.4.2 Deferral Solution

The Deferral solution design comprises multiple smart contracts representing different approaches or solutions for decentralized referral systems. The first solution contract starts

with a simple design and is gradually enhanced with features for extending the contract's process evaluation or reward distribution logic. Thereon, all major changes inside a contract would lead to a new version of a Deferral solution contract. The versioning of the different contracts is based on changes made either from a logical perspective, such as when functionality was added or removed from the contract, or from an implementation standpoint, where the code was changed, but the underlying logic remained the same. This chapter does not cover the implementation and code changes between the different contract versions. These aspects are discussed and evaluated in Chapters 5 and 6, respectively. With this approach, each contract represents a distinct version of a solution design. Preserving the different contract versions is beneficial for the evaluation as it allows the different contracts to be assessed against one another.

The subsequent sections define the various solution designs and smart contract versions. The design and functionality are explained for each solution with an overview of the referral conditions and a solution classification according to Section 3.3.

4.4.2.1 Referral Payment Transmitters

The first set of solution contracts and their design is referred to as referral payment transmitters. In total, there are three versions of referral payment transmitter contracts. Version one of the payment transmitter smart contracts served as a proof of concept for the solution architecture and thus is kept simple. This contract's main and only function is to accept payment from referees and forward a proportion to the receiver and another to the referrer. The contract design is depicted in Figure 4.5. The functionality of forwarding or transmitting a payment to a receiver is essential and shared among all upcoming solution contracts. The exact amount of the payment, the amount of the referral reward, and the address of the payment receiver are stored on the contract. Besides that, there is no other data stored on the transmitter contract. The payment amount and referral reward are stored as absolute numbers. All these values are fixed and can only be adjusted by the contract owner. The payment amount must always be bigger than the referral reward amount. Referees have to send a referral payment transaction to the smart contract. In this case, the payment transaction must send the exact amount of a native cryptocurrency asset set on the contract *e.g.,* 5 ETH in the case of Ethereum, and include a valid address in the transaction data representing the user who referred them (*cf.* Figure 4.5). The address included in the transaction represents the referrer or the referrer's address.

Based on the incoming referral transactions, the assets are either forwarded as a payment to the receiver's address or sent to the referrer's address as a reward. Further, Figure 4.5 sequentially portrays how the referral payment transaction triggers the forwarding of the payment and distribution of the rewards always within the same transaction execution (*cf.* [1] marked red in Figure 4.5).

Referral Conditions: The referral conditions for the referral payment transmitter solution design are very clear. Referees must send a referral payment transaction to the contract, including a valid referrer address and the exact amount of cryptocurrency asset required and defined by the contract. If the transaction succeeds, the referral conditions are fulfilled, and a predefined proportion of the payment amount is sent to the referrer's

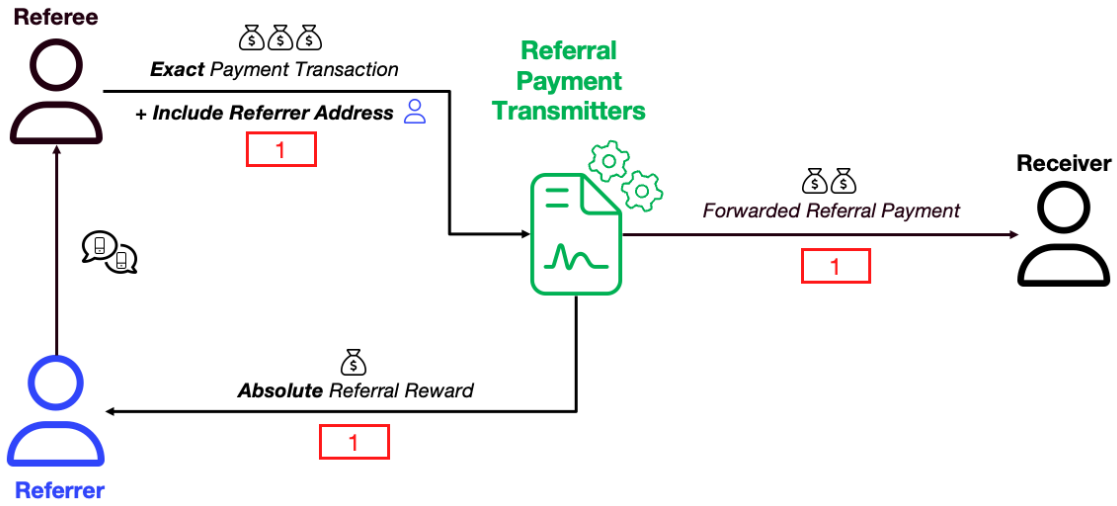


Figure 4.5: Solution Design for the Referral Payment Transmitter Architecture

address as a referral reward. The rest of the payment amount is forwarded to the receiver's address. Hence the referral system contract operates as a trivial payment proxy or transmitter. This kind of logic is used as a base and included in some form in all solution designs that is discussed in the upcoming sections.

Referral System Classification: The referral payment transmitter solution has no restriction regarding qualified users that can participate in the referral system *e.g.,* every valid address can be used as a referrer address. The allocation of the rewards is one-sided, as only the referrer receives a proportion of the payment. Lastly, the reward distribution only happens on one level as only one referrer receives rewards, and no referral chain is formed, including multiple levels of referrer addresses.

Moreover, there is a second and third version of payment transmitter contracts. These contracts have the same logic and functionality as version one but incorporate an upgradable pattern [97]. Detailed information about the upgradable pattern and its implementation can be found in Section 5.2.1. A key aspect of these upgradable contracts from a design point of view is their ability to allow the contract owner to modify and update the contract's implementation post-deployment, unlike regular smart contracts, which cannot be altered once deployed.

Consequently, this introduces a certain level of centralization to the contracts and the referral system. With upgradable smart contracts, there is a trade-off between the flexibility and security provided, which enable fixing bugs or adapting referral conditions after the deployment, and the potential for misuse or changes that could negatively impact the referral systems functionality and its users. From here on, all contracts are implemented with the upgradable pattern, accepting a certain degree of centralization. Implementing security advantages primarily drove the decision to use upgradable smart contacts. However, all contracts could easily be adapted and implemented without an upgradable pattern, removing the accepted degree of centralization.

4.4.2.2 Referral Payment Quantity Evaluators

Extending on the referral payment transmitter design, the referral payment quantity solution introduces a few changes and extensions. Both of the referral payment quantity contracts are upgradable as well [97]. The contracts store the receiver address and the relative proportion of the payment amount distributed to the referrers as percentage values on the contract. Additionally, the contract stores the payment quantity threshold value. The payment quantity threshold value is used to evaluate the referral process.

The main difference between the payment transmitter and payment quantity contracts lies in the referral conditions or the evaluation of the referral process. As for the payment transmitters, every process was completed after one successful payment that sent an exact predefined amount of crypto assets. In this solution design, referral payment transactions no longer require sending a predefined amount of crypto assets. Instead, the referral processes are only complete if referees have made more payments as defined in the payment quantity threshold value. For instance, if the payment quantity threshold is two, the referee has completed the referral process with the third payment transaction. However, in this case, there are no requirements or limitations for the value sent within each payment. Within this transaction that exceeds the threshold value, the referral process is evaluated as complete, and the rewards are distributed to the referrer. In Figure 4.6, this is illustrated as the referral payment transaction, and forwarding of the payment always occurs in every referral transaction (*cf.* [1] in Figure 4.6). Nonetheless, the distribution of the referral rewards only happens if the referral process has been completed (*cf.* [2] in Figure 4.6).

In a scenario where the referral process is completed within the first referral payment transaction *e.g.,* the payment quantity threshold value is set to zero, the sequential flow of the referral process (*cf.* marked red in Figure 4.6) would be equal to the referral payment transmitter design discussed before (*cf.* marked red in Figure 4.5).

The rewards are calculated from the total and accumulated value the referee has sent over all the recorded transactions and the reward percentage value defined on the contract. For example, suppose the referee has sent a total of 600 ETH over three transactions and the payment value threshold is two, and the referral reward is set to 10%. In that case, the contract sends 60 ETH to the referrer's address within the third referral payment transaction (*cf.* Figure 4.6).

Therefore, the contract has to store additional data about the referral payment transactions of every user. The contract stores the corresponding referrer address provided for every referee's address during the first referral transaction. It always stores the referrer address sent within the initial payment transaction. Hence, if a referee changes the referrer's address during one of the later referral payment transactions, it does not affect the referral process. Additionally, it keeps track of the total payment value and quantity across all the transactions a particular referee executes. Referees cannot set their address as the referrer's, as the two must be distinct. Further, the contract stores if the referral process has already been completed for a particular referee address. Once a referee completes the referral process, their address becomes ineligible for sending further payment transactions to the referral payment quantity contract.

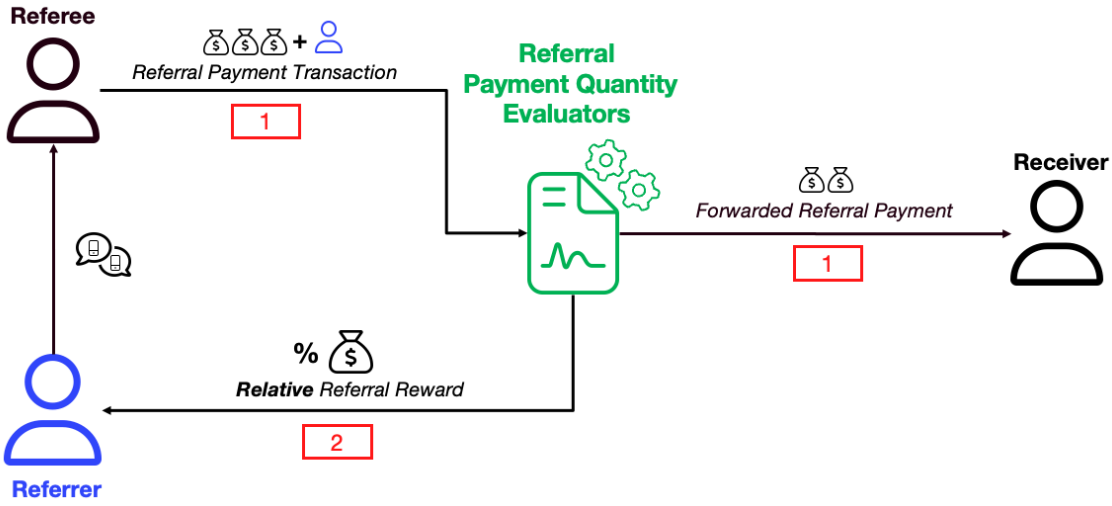


Figure 4.6: Solution Design for the Referral Payment Quantity Architecture

Referral Conditions: The referral conditions for the referral payment quantity contracts require the users *i.e.,* referees, to send $x + 1$ payment transactions where x is the payment quantity threshold set and stored on the contract. At this point, it is to say that the contract owner can update the value for the payment quantity threshold. As a result, the referral conditions can change during the referral process. From an implementation point of view, this has no effect as the contract and referral process still work correctly and as expected. However, the flexibility this provides to the program provider *i.e.,* contract owner, could confuse the user side. The ability to change this value could also be prevented, and the contract could be adapted. The resulting trade-off and discussion are similar to the argument about using upgradable contracts.

Referral System Classification: Regarding classification, the referral payment quantity contracts remain almost identical to the previous referral payment transmitters. The reward allocation is one-sided, and the distribution happens on one reward level. Regarding qualified users, there is one small difference: referees cannot refer themselves, which was not present in the first version of the referral payment transmitter contract. However, the third version of the referral payment transmitter has also already introduced this small limitation concerning qualified users.

4.4.2.3 Referral Payment Value Evaluators

The next category of solution designs is referred to as referral payment value evaluators. Especially the first version of the referral payment value contract is very similar to the previously discussed referral payment quantity contracts. They store the same values on the contract, with one exception. A payment value threshold value replaces the payment quantity threshold value. This payment value threshold is used uniformly for the referral process evaluation. The referees must send payment transactions with a total accumulated value exceeding the contract's defined payment value threshold. The referral process is completed with the first payment transaction, where the total payment value exceeds the

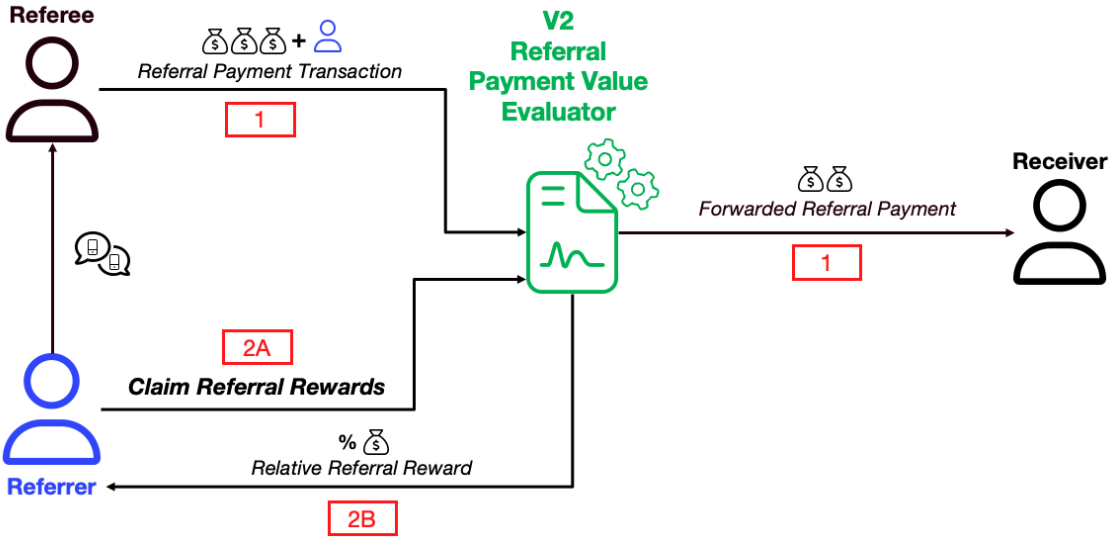


Figure 4.7: Solution Design for V2 of the Referral Payment Value Architecture

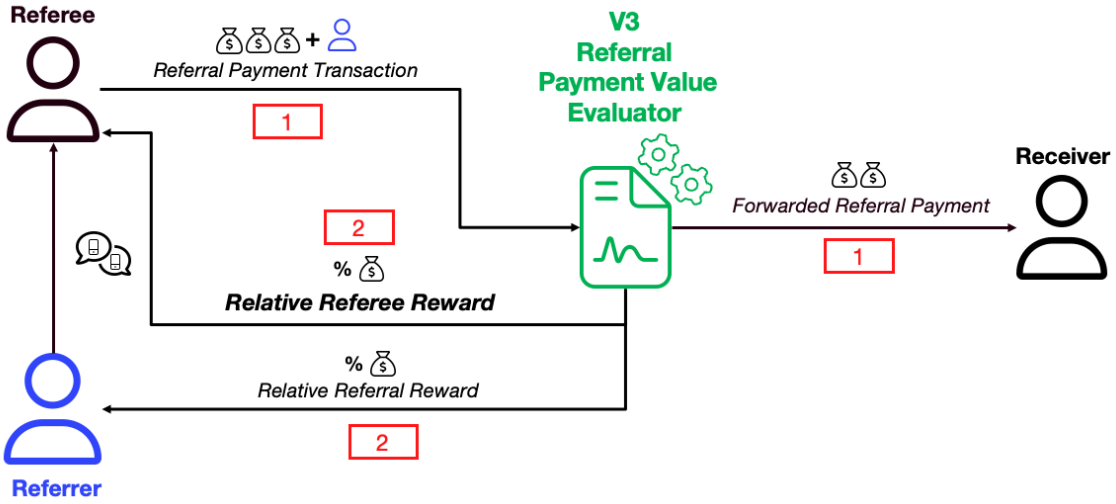


Figure 4.8: Solution Design for V3 of the Referral Payment Value Architecture

threshold value. Other than that, the data stored and the rest of the functionality are the same as described above for the referral payment quantity contracts (*cf.* Figure 4.6).

The second and the third referral payment value contracts differ slightly from the first version. In the second version, the referral rewards are not sent directly and within the same payment transaction that completes the referral process. Instead, the referral reward amount is stored for eligible referrer addresses on the contract. Hence if the referee has completed the process, the corresponding referrer can send a claim transaction to the contract to claim and receive the rewards (*cf.* [2A] and [2B] in Figure 4.7). This includes an additional step between the completion of the referral process and the distribution of the rewards. Thereby, the reward distribution is slowed down and must be initiated explicitly by the user. More about the effects and implications of this design change is discussed in Chapters 5 and 6.

The third version of the contract introduces a more flexible reward distribution system by incorporating a referee reward percentage. It allows both the referrer and the referee to benefit from the referral process. The contract stores a referee referral reward percentage value. This referee reward percentage value defines the proportion of the referral reward distributed to the referee. Hence, the referral reward is split between the referrer and the referee based on the specified referee reward percentage (*cf.* Figure 4.8). The contract calculates the reward for the referee by multiplying the referral reward with the referee reward percentage, and the remaining reward is then distributed to the referrer. For instance, if we assume a referee reward percentage value of 25% and take the example form above where the referral reward for the referrer is 60 ETH, the referee would receive 15 ETH, and the referrer would receive 45 ETH. Therefore the referral system ensures that both parties benefit from the referral rewards. Furthermore, by adjusting the referee reward percentage value, the contract can distribute rewards exclusively to the referrer when set to 0% or to the referee when set to 100%. This flexibility allows this contract version to adapt to different use cases and requirements.

Referral Conditions: The referral conditions are the same for all versions of the payment value contracts. Compared to the referral payment quantity contracts, the referral conditions require the total accumulated value of all payment transactions per referee to be greater than y, where y is the payment value threshold set on the contract.

Referral System Classification: Concerning the classification, versions one and two can be classified similarly to the previous solution contracts. They imply the same limitations on qualified users, one-sided reward allocation, and single-level reward distribution. Nevertheless, version three of the payment value evaluator contracts introduces the possibility of adjusting the reward allocation arbitrarily with the referee reward percentage value stored on the contract. Thus, the reward allocation can be either one-sided or two-sided.

4.4.2.4 Referral Payment Multilevel Reward Evaluators

The multilevel reward evaluator contracts combine referral process evaluation characteristics from the referral payment quantity and value contracts. Further, they introduce new functionality concerning reward distribution. This solution design combines the referral conditions from the payment quantity and payment value contracts. Hence, the contract stores both a payment quantity and a threshold value. To complete a referral process, a referee has to send a certain amount of payment transactions that, on the one hand, surpass the defined payment quantity threshold in terms of transactions sent and, on the other hand, exceed the payment value limit in terms of total accumulated value sent through the payments.

There are two versions of referral payment multilevel reward evaluator contracts. Both versions' main functionality is distributing rewards to the most recent referrer and multiple previous referrers. Hence, as the name implies, the referral rewards are distributed on multiple referral levels. For example, if user (A) is referred by user (B), who was in turn referred by user (C), upon completion of the referral process by user (A), both user (B) (referrer of user A) and user (C) (referrer of user B) receive a portion of the referral rewards (*cf.* Figure 4.9). This creates a referral chain including all prior referrers, with rewards

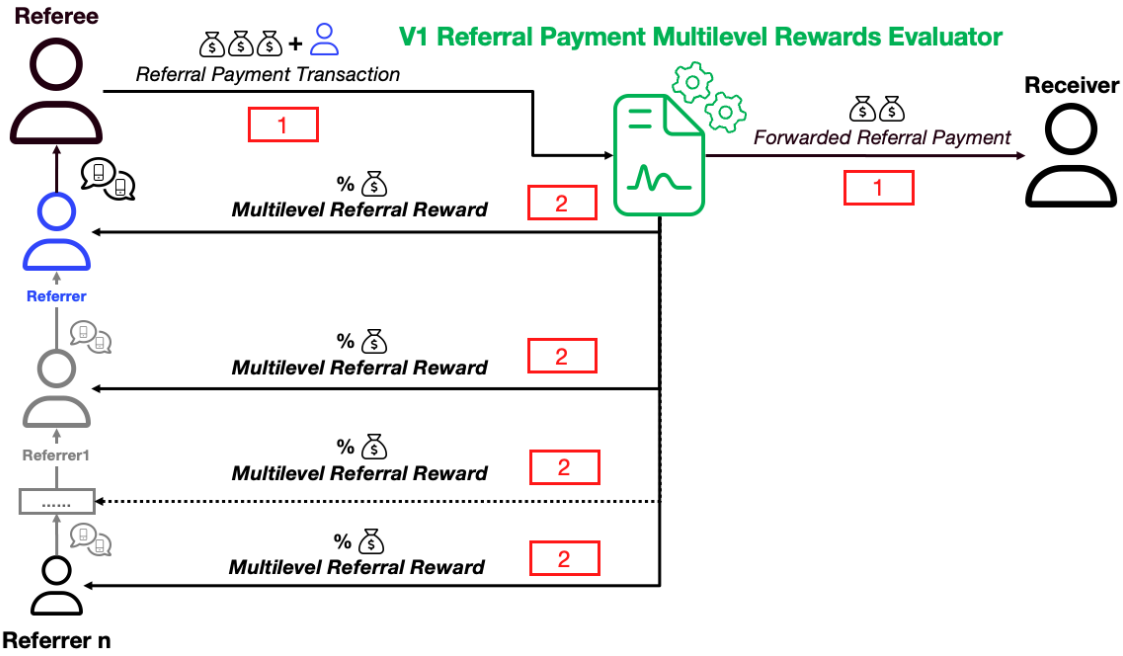


Figure 4.9: Solution Design for V1 of the Referral Payment Multilevel Rewards Architecture

distributed along the chain. Although different reward distributions could be employed for each level, the rewards are equally split among all eligible referrers. Consequently, if the referral reward in this example is 60 ETH, both users (B) and (C) would receive 30 ETH each.

In the case of the first version of the multilevel rewards contract, there is no restriction on the length of the referral chain. As a result, rewards are distributed to all previous referrers up until the initial *i.e.,* root referrer. Especially with very large referral chains, this can become problematic and inefficient. A detailed evaluation of the implications of this design choice and the impact of long referral chains can be found in Chapter 6.

The second version of the multilevel rewards contracts incorporates two additional features. Firstly, it reintroduces a referee reward percentage value, enabling two-sided referral rewards similar to the third version of the referral payment value contracts (*cf.* Section 4.4.2.3). Secondly, it includes a maximum reward level value stored on the contract. This value determines the maximum length of the referral chain for reward distribution *i.e.,* to how many previous referrers the rewards are distributed (*cf.* Figure 4.10).

In an exemplary scenario, user (1) is referred by user (2), who is then referred by user (3), who is, in turn, referred by user (4), and so on. The maximum reward level value on the contract is set to three. If user (1) completes the referral process, the referral rewards are distributed to the three nearest referrers of user (1) up along the referral chain. Assuming the total reward is 60 ETH and all of the rewards are distributed to the referrers *i.e.,* the referee reward percentage is set to 0%, user (2), user (3), and user (4), would each receive a reward of 20 ETH. If the maximum reward level is six, starting from user (2) to user (7), every referrer would receive 10 ETH.

If, in the same example, the referral contract has defined a referee reward percentage value of greater than0%, *e.g.,* 50%, 30 ETH would be sent to the referee who completed the

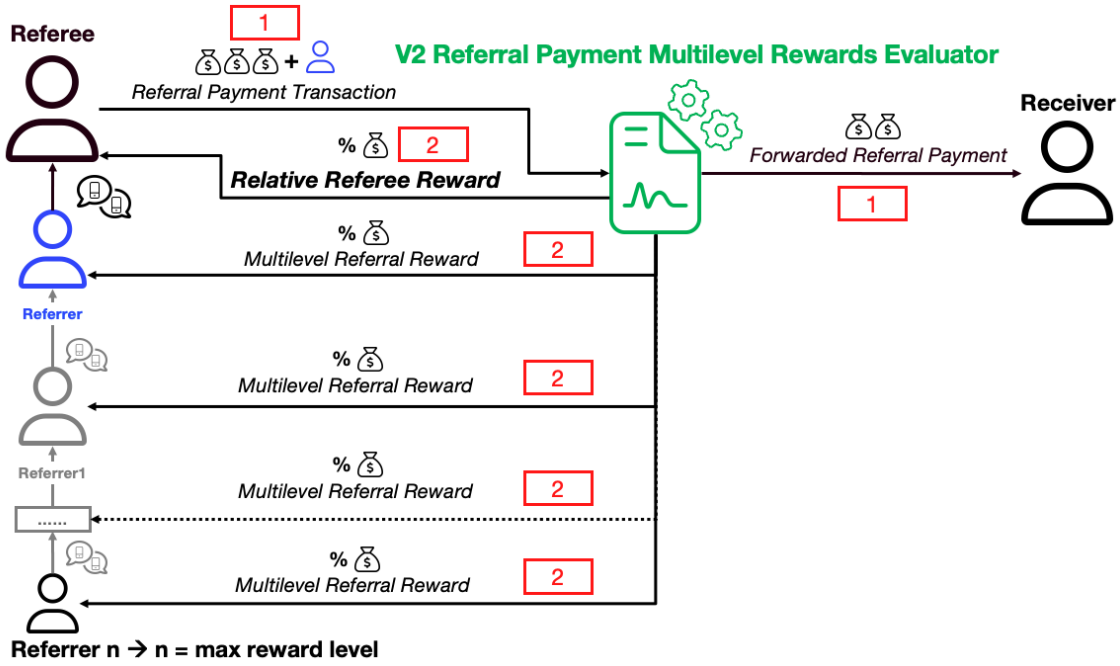


Figure 4.10: Solution Design for V2 of the Referral Payment Multilevel Rewards Architecture

process. The other 30 ETH would be distributed equally to the eligible referrers along the referral chain (*cf.* Figure 4.10).

Referral Conditions: All versions of the referral payment multilevel reward contract require the referees to send $x + 1$ payment transactions where x is the payment quantity. Additionally, the total accumulated value of all payment transactions per referee must be greater than y, where y is the payment value threshold set on the contract. Thus, the referral conditions combine the previous referral payment quantity and referral payment value contract versions.

Referral System Classification: When examining the two multilevel reward contracts, it is evident that they can be categorized as multilevel due to their distinct reward distribution. Regarding reward allocation, version one permits only one-sided rewards, while version two enables the distribution of two-sided rewards. The contract owner is responsible for determining and setting the reward split.

Both multilevel reward contracts incorporate an additional constraint concerning qualified users. Previously, the sole restriction was that referrers could not refer themselves, meaning the referrer's address could not match the referee's address. This limitation remains in place for these contracts. However, an additional restriction requires every referrer to be a registered user who has done a payment transaction before *i.e.,* a customer. This specification dictates that every referrer must have completed a referral payment transaction, registering their address on the contract without a corresponding referrer address. If an address is registered on the contract without a referrer, it is considered a root referrer. Hence, sending referral payment transactions to the contracts is now possible without a referrer address to enable this functionality. For this kind of regular payment transaction, all the sent assets are forwarded to the receiver, and the sender is registered as a root referrer. Consequently, the multilevel rewards contracts impose a stricter criterion

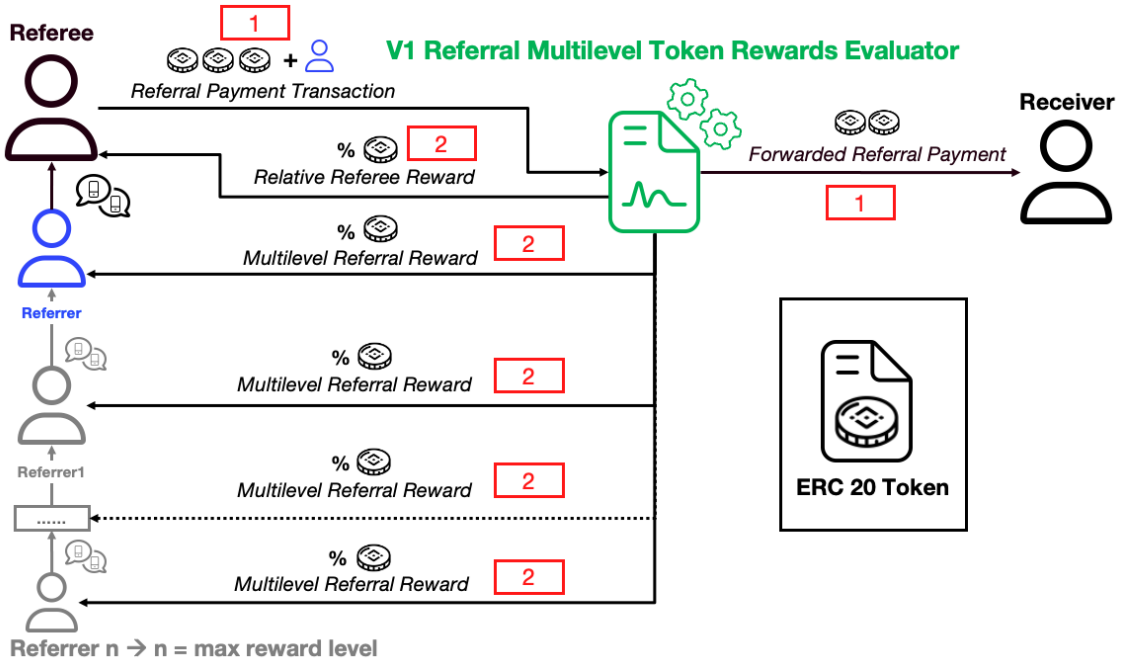


Figure 4.11: Solution Design for V1 of the Referral Token Multilevel Rewards Architecture

on qualified users who can become referrers. Only customers *i.e.,* registered users, are permitted to serve as referrers.

4.4.2.5 Referral Payment Multilevel Token Reward Evaluators

The final solution contract incorporates multilevel token rewards. It closely resembles version two of the multilevel rewards contract. The primary distinction is that with this design, an arbitrary ERC20 token can now be utilized as a cryptocurrency for referral payments instead of native cryptocurrency assets. Consequently, both the forwarded payment and referral rewards are sent, received, and distributed in the form of an ERC20 token (*cf.* Figure 4.11). The multilevel token reward contract permits and accepts only one predefined token. This ERC20 token must be stored on the contract. As opposed to all previous solution designs updating the values used for evaluation of the referral process *e.g.,* payment quantity or value thresholds, the ERC20 currency token stored on the contract cannot be updated after the initial creation of the referral contract. Changing the currency token for ongoing referral processes and referral payments would lead to inconsistencies in the process evaluations and reward distribution. Apart from this difference, the multilevel token reward design remains consistent with the previous solution.

Introducing the possibility of using an ERC20 token as the currency of the referral system opens doors to additional use cases of such a system. For instance, multiple different companies *i.e.,* their referral systems could use the same ERC20 token as a reward. Thereby users could earn the same rewards across multiple referral systems, which would create a shared referral reward ecosystem. Such an ecosystem could open new doors for further marketing activities. For instance, rewards in the form of ERC20 could be integrated into the DeFi space, making the tokens tradable on a DEX.

Referral Conditions: The referral conditions outlined for the token multilevel reward contract remain consistent with those previously detailed in Section 4.4.2.4.

Referral System Classification: Also, the classification stays the same compared to version two of the multilevel reward contracts. The same restrictions apply to qualified users. Rewards are allocated two-sided and are distributed on multiple levels.

4.5 Evaluation Metrics and Measures

Various similar and continuously extended solution designs have been introduced in the previous sections. With multiple solution designs, it is crucial to have metrics and measures in place that can be used to evaluate and compare the contracts against each other. These metrics can reveal benefits or inefficiencies about the different design choices for the referral evaluator contracts. Consequently, this section discusses the evaluation measures that are important to consider within this thesis. Broadly, the significant factors fall into three primary categories, cost, performance, and security. A detailed discussion on the specific metrics employed for evaluating these categories, their application, and measurement methods is conducted in Chapter 6.

4.5.1 Costs

Cost evaluation is fundamental for all the presented solution designs. The decentralized referral systems in the form of smart contracts involve financial assets and require resources to execute referral payment transactions as well as to provide and deploy the referral contract. Hence, the costs of the referral systems can be evaluated from a user and a provider perspective. Moreover, transactions on the blockchain unavoidably cause costs. Different design decisions and implementations of the referral contracts initiate varying costs. Thus, the transaction or deployment costs can help to identify the most economically viable solution design.

4.5.2 Performance and Scalability

Consequently, the solutions must be analyzed in terms of performance and scalability. Examining how the various contracts manage high volumes of participating users and the resulting influx of referral payment transactions is vital. The analysis should consider how the costs evolve and respond to the increased load and help identify bottlenecks and areas for optimization of the solution design

4.5.3 Security

Conclusively, the security of each of the solutions cannot be overlooked. The implementation of the contracts should be inspected to identify potential vulnerabilities and attack vectors, ensuring the selected solution's resilience against hacks and malicious actors.

Chapter 5

Deferral - Implementation

This chapter dives into the implementation of the Deferral solution. All the different solution designs and smart contracts outlined in Chapter 4 are discussed. To start with, the development environment and setup, together with the solution technology stack and the repository structure, are introduced. Afterwards, the Deferral smart contracts code and implementation are illustrated and discussed. Lastly, other aspects of the final solutions, including all the tests, scripts, and other source code that has been developed, are presented.

5.1 Development Environment and Setup

The Deferral solution implemented as part of this thesis is organized and collected within two GitHub Repositories. The two repositories contain all the implemented source code and corresponding files relevant to the thesis. The main Deferral repository [132] entails all the solution smart contracts, their tests, as well as various scripts for the deployment and evaluation of the contracts and their results.

The second Deferral Visualization GitHub repository [133], is a submodule or subrepository of the main Deferral repository and includes all relevant scripts used for analyzing and visualizing the generated results of the Deferral evaluation processes.

For more general information or details about the implementation, please refer directly to the two repositories [132, 133] and their respective `README.md` files.

5.1.1 Solution Technology Stack

The following sections introduce the main technologies and solutions that have been used during the implementation. In part, it outlines the reasoning behind the technology decisions that have been made. For a complete overview of all the technologies used, please refer to the GitHub repositories [132, 133].

5.1.1.1 Development Environment and Tools

Among others, the following technologies or tools have been used to support the development of the Deferral Solution during the implementation phase:

- **Hardhat**: The Hardhat development environment has been used as it facilitates building, testing, and deploying smart contracts onto different networks, including local or test networks [61]. It provides Hardhat scripts for task automation and a local Hardhat network that facilitates contract testing.
- **TypeScript**: The majority of the implemented code is written in TypeScript [135], enabling a better development experience and maintainability while improving code quality.
- **Yarn**: In combination with Hardhat, the yarn package manager [151] has been used to integrate executable scripts that would allow for easier and quicker deployment, evaluation, and testing of the developed smart contracts.
- **Chat GPT**: The Chat GPT AI model [94] has been used to support the implementation process by offering code suggestions, debugging assistance, producing documentation, or answering technical questions.

5.1.1.2 Smart Contracts

For the Deferral solution smart contracts, the following technologies or tools have been used:

- **Solidity**: All the smart contracts have been written in Solidity [127]. This allowed for implementing EVM-compatible smart contracts that can be used and tested on multiple blockchains within the broad and well-established Ethereum ecosystem.
- **Solcover**: Solcover [129] is a code coverage tool for Solidity contracts that enables measuring and testing coverage for smart contracts.
- **TypeChain**: In combination with Hardhat [61], TypeScript [135] and Solidity [127] TypeChain [134] has been used. TypeChain helps to create TypeScript bindings for Ethereum smart contracts, providing type-safe interactions and reducing the risk of errors.

5.1.1.3 Data Analysis and Visualization

Within the Deferral Visualizations sub-repository [133], the subsequent technologies have been utilized:

- **Python**: For analyzing and visualizing the result data, Python has been used[109].
- **Plotly**: The Plotly graphing library [104] for Python has been used to create visualizations.

5.1.1.4 Code Quality and Linters

Eventually, the following supportive tools have ensured code quality during the development of all the source code:

- **ESLint**: EsLint [52] offers linting utilities that help maintain code quality and enforce coding standards.
- **Solhint**: Solhint [107] is a linter and code style checker for Solidity code, ensuring smart contract code quality and adherence to best practices.
- **Prettier**: The Prettier code formatter [106] enforces consistent style across code source files, improving readability and maintainability.

5.1.2 Continuous Integration

Within the Deferral repository [132], a Continuous Integration (*CI*) workflow is set up using GitHub Actions [57] to run various checks and tests on the codebase automatically. The CI job includes the following steps:

- Checkout the repository, install Node.js [93] and all the dependencies.
- Lint the code, compile the contracts and generate TypeChain [134] bindings
- Run all the Hardhat [61] smart contract tests and report the Solidity code coverage.

The workflow is defined in the `.github/workflows/test.yml` file and executed on workflow dispatch, pull requests, and pushes to the main branch in the Deferral repository. This CI automation adds another layer of code quality and maintainability as all the code that lies on the main branch complies with the code quality checks and smart contract tests.

5.1.3 Repository and Folder Structure

To wrap up the development environment and setup, the overall project or folder structure of the Deferral repository is broadly outlined to provide a better overview of the different parts of the implementation. Again, only the most relevant folders are listed here. For a complete overview, the GitHub repositories can be considered [132, 133]. The Deferral project is structured as follows:

```
/
├── contracts/:  all solution smart contracts
│   ├── referral-evaluators/
```

```
    ├── referral-multilevel-token-rewards/
    ├── referral-payment-multilevel-rewards/
    ├── referral-payment-quantity/
    ├── referral-payment-transmitter/
    └── referral-payment-value/
  └── .../
├── helpers/:  helper functions for testing, deployment, & evaluation of contracts
├── logs/:  stores generated log files of deployments & evaluations of contracts
  ├── deployments/
  └── evaluations/
├── results/:  result data & visualizations used for the evaluation in Chap-
  ter 6
├── scripts/:  scripts for testing, deploying & evaluating contracts
  ├── deployment/
  │  └── .../
  └── evaluation/
     └── .../
├── test/:  all tests for the smart contracts grouped by their design
  ├── referral-multilevel-token-rewards/
  ├── referral-payment-multilevel-rewards/
  ├── referral-payment-quantity/
  ├── referral-payment-transmitter/
  └── referral-payment-value/
├── .../
└── visualizations-deferral/:  submodule for visualization & analysis code
```

5.2 Referral Evaluator Smart Contracts

Within the upcoming sections, the main components of the Deferral solution implementation, the prototype referral evaluator smart contracts, are reviewed and discussed. Similar to Chapter 4 the multiple contracts are grouped by their underlying design. However, at this point, the focus lies more on their technical implementation and the Solidity code.

For each of the contracts, a Unified Modeling Language (*UML*) class diagram has been generated [126]. The class diagrams visually represent the contracts' structures, variables, and functions combined with other features of the Solidity code. In this context, a `generate_contract_diagrams.sh` shell script has been implemented to generate and store UML visualizations of each contract automatically. The script can be found in the Deferral repository [132]. Furthermore, for other excerpts or snippets of the Solidity code that are shown in these sections, the complete source code can be found in the Deferral repository as well. Further, examples of larger code listings showing details about the implementation of the Solidity contracts can be found in the Appendix in Section C.

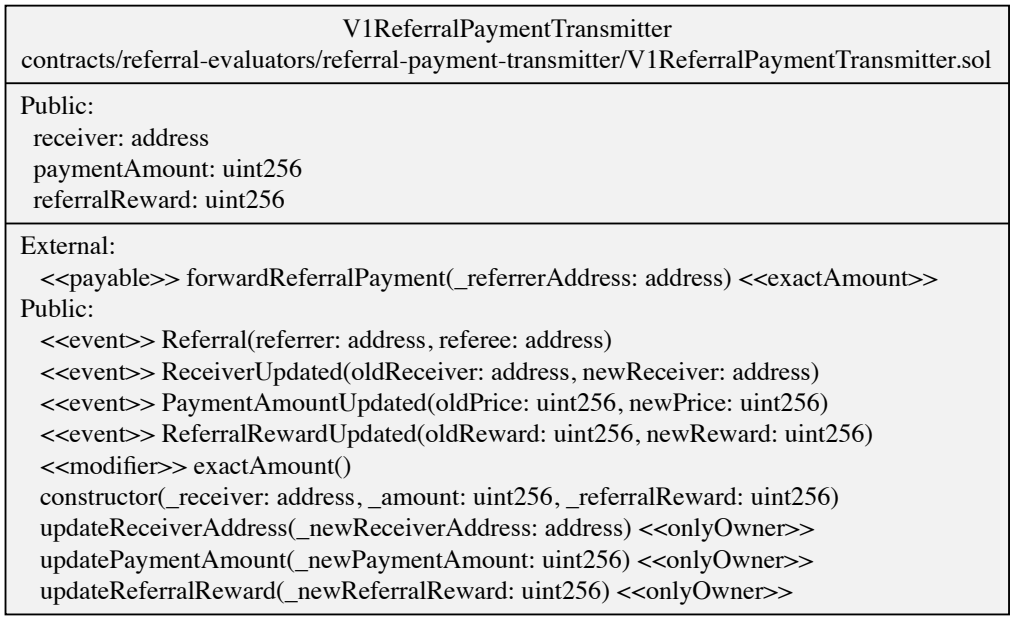


Figure 5.1: UML Class Diagram of the V1ReferralPaymentTransmitter Contract

5.2.1 Referral Payment Transmitter Contracts

The referral payment transmitter referral contracts implement the simplest solution for a decentralized referral system prototype as outlined in Section 4.4.2.1.

5.2.1.1 V1ReferralPaymentTransmitter

Figure 5.1 illustrates the implementation of the V1ReferralPaymentTransmitter smart contract. It shows the three different variables that are stored on the contract. The `paymentAmount` value defines the exact value that must be sent within every referral payment transaction. Every referral transaction has to call the `forwardReferralPayment` function and send the correct amount of native cryptocurrency assets. A valid `_referrerAddress` argument must be included for the function call. The forwardReferralPayment function is set as `external` *i.e.,* can only be called by other contracts or accounts and accepts native cryptocurrency assets as indicated by the `payable` keyword (*cf.* Figure 5.1).

```
1  contract V1ReferralPaymentTransmitter is Ownable {
2      [...]
3      // modifier to guarantee the exact amount
4      modifier exactAmount() {
5          require(
6              msg.value == paymentAmount,
7              "tx must send exact payment amount"
8          );
9          _;
10     }
11     [...]
12 }
```

Listing 5.1: `exactAmount` Modifier of V1ReferralPaymentTransmitter

Furthermore, the function accepts payments only if they match the exact amount of assets specified in the `paymentAmount` value stored on the contract. A custom `exactAmount` function modifier has been defined to achieve this. The implementation of this modifier and its application in the `forwardReferralPayment` function can be observed in Listings 5.1 and 5.2 respectively. The code excerpts also show that the V1ReferralPaymentTransmitter contract utilizes the `Ownable` contract template by OpenZeppelin [95]. The `Ownable` template provides simple access control for the contract by defining a single owner of the contract restricting certain `public` functions (*cf.* Figure 5.1) that can update values stored on the contract to be performed only by the owner.

Moreover, the `referralReward` value defines the proportion of the `paymentAmount` that is sent to the `_referrerAddress` account. Therein, the code enforces the `referralReward` always to be smaller than the `paymentAmount` as it can be observed in Listings C.1 and C.2. The `receiver` address stored on the contract specifies the target account for receiving the referral payments.

```
1  contract V1ReferralPaymentTransmitter is Ownable {
2      [...]
3
4       // forward paymentAmount to the receiver and send referralReward to
           ↪ the referrerAddress
5      function forwardReferralPayment(
6          address payable _referrerAddress
7      ) external payable exactAmount {
8          uint256 receiverAmount = msg.value - referralReward;
9          uint256 referrerRewardAmount = msg.value - receiverAmount;
10         // forward payment to receiver
11         receiver.transfer(receiverAmount);
12         // send referral rewards to referrer
13         _referrerAddress.transfer(referrerRewardAmount);
14         emit Referral(_referrerAddress, msg.sender);
15     }
16
17     [...]
18 }
```

Listing 5.2: `forwardReferralPayment` Function of V1ReferralPaymentTransmitter

Eventually, Figure 5.1 also depicts the various `events` that are used and emitted in the V1ReferralPaymentTransmitter contract. The names of the different `events` imply on which occasion they are emitted. For instance, as can be seen in Listing 5.2, the `Referral event` is emitted every time a referral process *i.e.,* referral payment is completed. The primary purpose of `events` is to log information about the happenings of the V1ReferralPaymentTransmitter contract to the blockchain. Nonetheless, the `events` defined in this as well as all the other smart contracts implemented, have not been further utilized in this thesis.

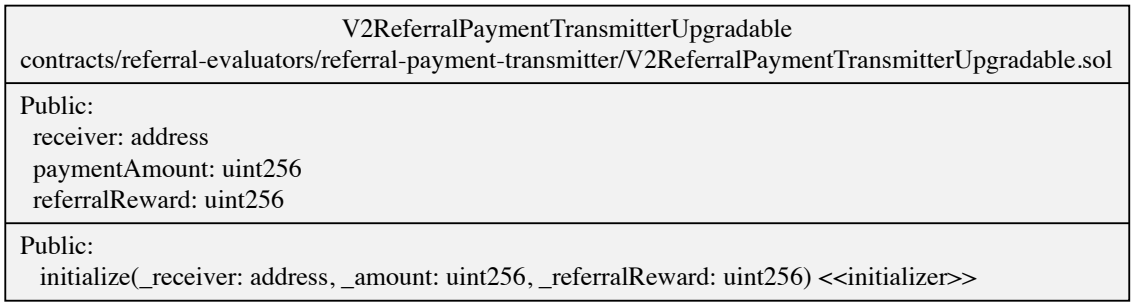


Figure 5.2: UML Class Diagram of the V2ReferralPaymentTransmitterUpgradable Contract

5.2.1.2 V2ReferralPaymentTransmitterUpgradable

As discussed in Section 4.4.2.1, the V2ReferralPaymentTransmitterUpgradable implements the upgrades pattern by OpenZeppelin [97]. To do so, the contract utilizes the `Initializable` template [96]. This template allows secure and seamless improvements to smart contracts that have already been deployed to the blockchain.

```
1  contract V2ReferralPaymentTransmitterUpgradable is Initializable,
       ↪ OwnableUpgradeable {
2      [...]
3      // upgradable-contracts initializer --> set receiver address and
           ↪ referral conditions
4      function initialize(
5          address payable _receiver,
6          uint256 _amount,
7          uint256 _referralReward
8      ) public initializer {
9          // set owner
10         __Ownable_init();
11         require(
12             _amount > _referralReward,
13             "reward must be portion of paymentAmount"
14         );
15         receiver = _receiver;
16         paymentAmount = _amount;
17         referralReward = _referralReward;
18     }
19     [...]
20 }
```

Listing 5.3: `initialize` Function of V2ReferralPaymentTransmitterUpgradable

In comparison to the first version contract, the V2ReferralPaymentTransmitterUpgradable has no `constructor` method (*cf.* Listing C.1). Instead, in order to implement the upgradable pattern, it includes an `initialize` function (*cf.* Figures 5.1 and 5.2). Inside the `initialize` method, the contract variables that are stored on the `proxy` contract are set, and the `OwnableUpgradeable` template is initialized, as can be observed in Listing 5.3. The `OwnableUpgradeable` provides the same functionality as the `Ownable` template implemented in the V1ReferralPaymentTransmitter contract but adjusted to work with the `Initializable` template.

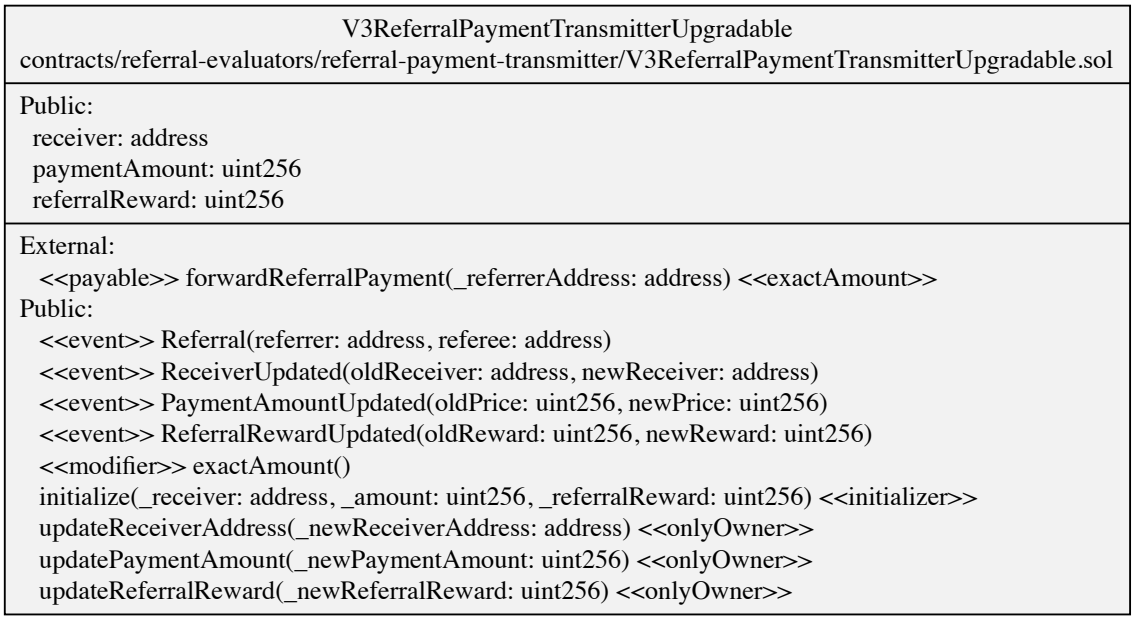


Figure 5.3: UML Class Diagram of the V3ReferralPaymentTransmitterUpgradable Contract

When deploying an upgradable or initializable smart contract, a `proxy` and `implementation` contract are deployed to work together and enable upgradeability. Therein, the `proxy` contract stores the state variables and delegates all function calls to the `implementation` contract, which contains the contract logic. Once an upgrade is needed, the `proxy`'s reference value to the `implementation` contract is updated. After the update, the `proxy` contract points and delegates all calls to a new implementation contract. In this way, the stored state is preserved in the `proxy` contract, while the logic can be upgraded in a new `implementation` contract.

The primary purpose of the V2ReferralPaymentTransmitterUpgradable implementation was to test the upgradable pattern. Hence, it misses the logic required to provide a decentralized referral system. This functionality is incorporated again in the third version of the referral payment transmitter contracts.

5.2.1.3 V3ReferralPaymentTransmitterUpgradable

The V3ReferralPaymentTransmitterUpgradable contract (*cf.* Figure 5.3) replicates the logic and referral process of the V1ReferralPaymentTransmitter, as described in Section 5.2.1.1. It also implements the upgradable pattern introduced in the previous section for the V2ReferralPaymentTransmitterUpgradable contract (*cf.* Listing 5.3). After all, it presents the final and upgradable version of the referral system implementing the referral payment transmitter design. The complete implementation of this contract can be found in the Deferral main repository [132].

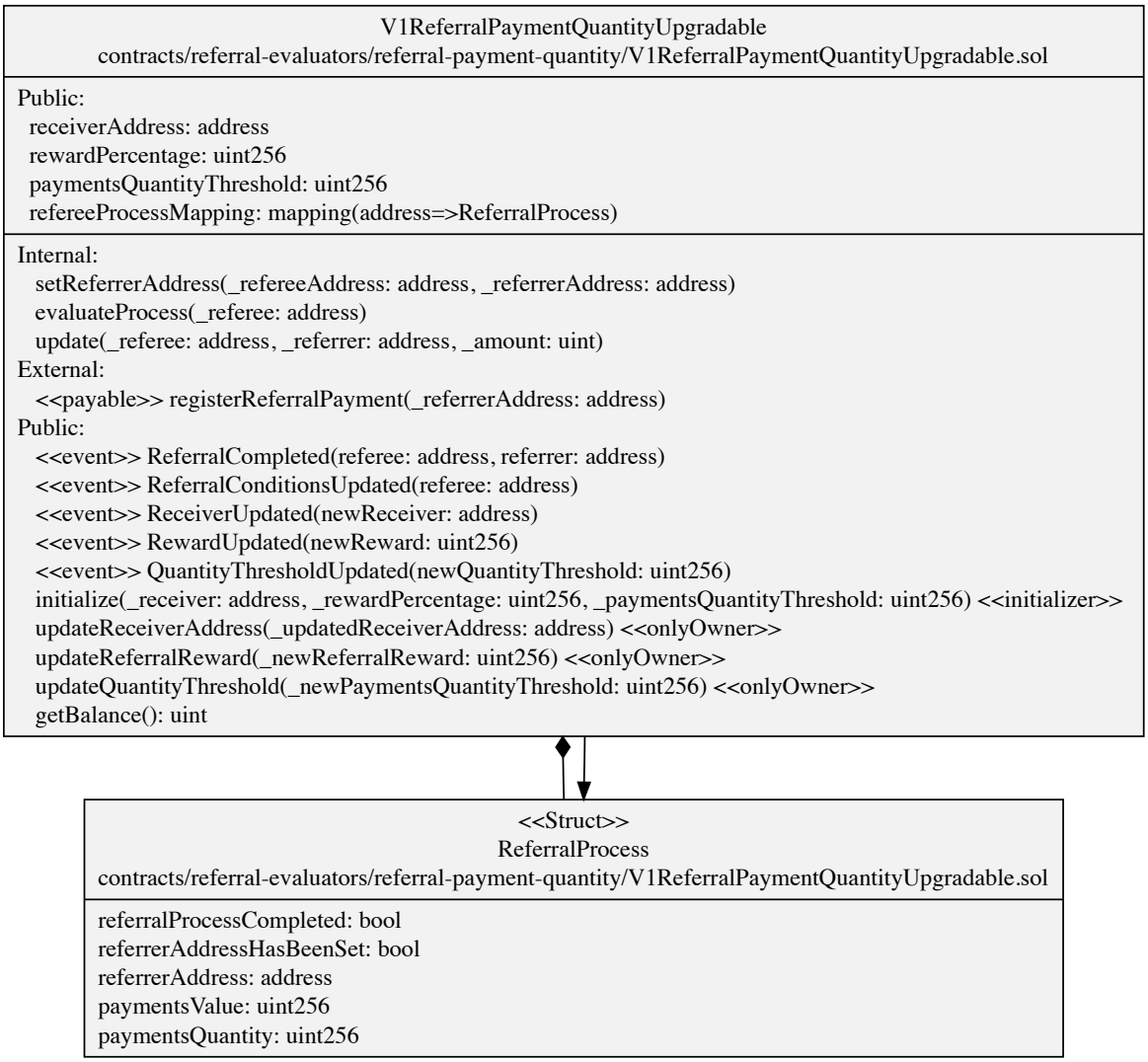


Figure 5.4: UML Class Diagram of the V1ReferralPaymentQuantityUpgradable Contract

5.2.2 Referral Payment Quantity Evaluator Contracts

As discussed in Section 4.4.2.2, the referral payment quantity evaluator contracts introduce an extended referral system applying a more complex evaluation of the referral process. Both versions entail the same logic and functionality. Nevertheless, they differ in their implementation.

5.2.2.1 V1ReferralPaymentQuantityUpgradable

The components of the V1ReferralPaymentQuantityUpgradable contract implementation are shown in Figure 5.4. Part of it is the `registerReferralPayment` method, which accepts and registers referral payments. As can be perceived in the corresponding UML class diagram (*cf.* Figure 5.4), this function no more implements a custom modifier for the exact amount of native assets that must be sent within the referral transaction.

As the V1ReferralPaymentQuantityUpgradable contract is designed to evaluate the referral process of referees by the number *i.e.,* quantity of referral payment transactions a referee has completed, it stores a `mapping` named `refereeProcessMapping` (*cf.* Listing 5.4). In this `mapping`, every referee `address` is mapped to a `ReferralProcess struct`.

```
contract V1ReferralPaymentQuantityUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    // mapping for referees including their data for referral conditions
        ↪ progress
    mapping(address => ReferralProcess) public refereeProcessMapping;
    [...]
}
```

Listing 5.4: `refereeProcessMapping` of the V1ReferralPaymentQuantityUpgradable

The `ReferralProcess struct` is a custom data type that allows to store and manage multiple variables related to the referral process of every referee under a single entity (*cf.* Figure 5.4). Listing 5.5 outlines the definition of the `ReferralProcess struct`. The five different `struct` values stored within the `refereeProcessMapping` are required for the contract to evaluate the referral process. In the course of every incoming referral payment transaction, these values are updated by the `internal update` function before they are evaluated to see if the process has been completed.

```
contract V1ReferralPaymentQuantityUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    struct ReferralProcess {
        // set true if the referral has been successful & rewards have been
            ↪ paid out
        bool referralProcessCompleted;
        // set true if the referrer address has been set
        bool referrerAddressHasBeenSet;
        address payable referrerAddress;
        uint256 paymentsValue;
        uint256 paymentsQuantity;
    }
    [...]
}
```

Listing 5.5: `ReferralProcess Struct` of V1ReferralPaymentQuantityUpgradable

A referee's referral process is evaluated within the `evaluateProcess` method. Listing 5.6 indicates how the referral process data is evaluated in the `evaluateProcess` method. Moreover, Listing 5.7 shows the `registerReferralPayment` function of the contract and how the `evaluateProcess` method is called in every referral transaction after the process data has been updated.

```
contract V1ReferralPaymentQuantityUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    function evaluateProcess(address _referee) internal {
        ReferralProcess storage currentProcess = refereeProcessMapping[
            _referee
```

```
        ];
        // require referrer address has been set
        require(
            currentProcess.referrerAddressHasBeenSet,
            "Referrer Address has not been set for this referee"
        );

        if (currentProcess.paymentsQuantity > paymentsQuantityThreshold) {
            uint256 calculatedReward = (currentProcess.paymentsValue / 100)
                ↪ *
                rewardPercentage;
            require(
                address(this).balance >= calculatedReward,
                "Contract has not enough funds to pay rewards"
            );
            currentProcess.referrerAddress.transfer(calculatedReward);
            currentProcess.referralProcessCompleted = true;
            emit ReferralCompleted(_referee, currentProcess.referrerAddress
                ↪ );
        }
    }
    [...]
}
```

Listing 5.6: `evaluateProcess` Function of V1ReferralPaymentQuantityUpgradable

Besides the `evaluateProcess` function, there are two other `internal` methods implemented in the V1ReferralPaymentQuantityUpgradable contract (*cf.* Figure 5.4). `Internal` methods can only be called within the contract itself.

```
contract V1ReferralPaymentQuantityUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    // register & forward payment and update referral process data
    function registerReferralPayment(
        address payable _referrerAddress
    ) external payable {
        require(msg.sender != _referrerAddress, "Sender cannot be referrer"
            ↪ );
        // get current referee process data
        ReferralProcess storage currentProcess = refereeProcessMapping[
            msg.sender
        ];

        // referral process must not be completed
        require(
            !currentProcess.referralProcessCompleted,
            "Referral process has been completed for this address"
        );
        // update progress
        update(msg.sender, _referrerAddress, msg.value);
        // calculate reward and payment prices
        uint256 reward = (msg.value / 100) * rewardPercentage;
        uint256 receiverAmount = msg.value - reward;
        // evaluate referral progress and if complete payout rewards
        evaluateProcess(msg.sender);
```

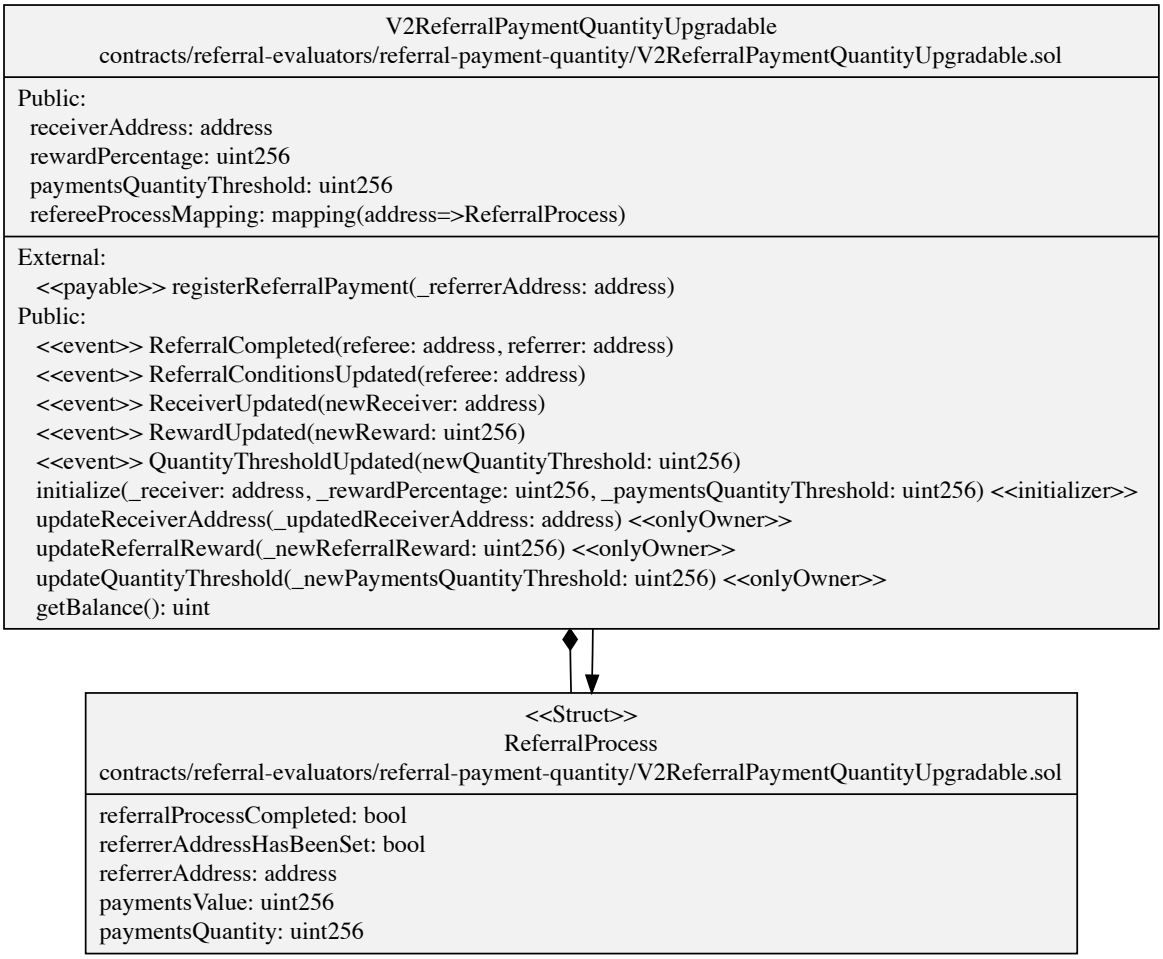


Figure 5.5: UML Class Diagram of the V2ReferralPaymentQuantityUpgradable Contract

```
25          // forward payment to receiver
26          receiverAddress.transfer(receiverAmount);
27      }
28      [...]
29  }
```

Listing 5.7: `registerReferralPayment` Function of V1ReferralPaymentQuantityUpgradable

As covered in Section 4.4.2.2, a referral process is completed if the `paymentsQuantity` value stored in the `ReferralProcess struct` exceeds the `paymentsQuantityThreshold` stored on the contract. Therein, the `evaluateProcess` method calculates the referral reward amount based on the total `paymentValue`, which has been continuously stored and updated in the `ReferralProcess struct` of the corresponding referee (*cf.* Listing 5.6). To calculate the eventual rewards, the `rewardPercentage` value that is stored on the contract is multiplied by the total `paymentValue` of the `ReferralProcess struct`. In addition, the `rewardPercentage` is restricted by the contract's implementation to represent a percentage value as can be seen in the `initialize` and `updateReferralReward` and `initialize` functions in Listings C.3 and C.4. Lastly, as part of the `evaluateProcess` method, the calculated reward is sent to the corresponding referrer of the referee who has completed the process, and the referral process is marked as completed (*cf.* Listing 5.6).

5.2.2.2 V2ReferralPaymentQuantityUpgradable

The V2ReferralPaymentQuantityUpgradable contract implements the same logic and functionality as version one of the referral payment quantity contracts. However, the UML class diagram in Figure 5.5 delineates that instead of splitting up the functionality in separate and `internal` functions as it has been done in version one, the contract keeps all the logic in fewer but therefore, larger functions. The `registerReferralPayment` function of the V2ReferralPaymentQuantityUpgradable contract shown in in the Appendix in Listing C.5 exemplifies this.

While this approach can negatively affect the readability of the contract, especially with larger and more complex code, it can have benefits regarding costs. The effects on the costs of this approach for implementing the V2ReferralPaymentQuantityUpgradable contract are discussed in the upcoming Chapter 6.

5.2.3 Referral Payment Value Evaluator Contracts

The referral payment value evaluator contracts follow a similar design as the referral payment quantity evaluator contracts discussed before (*cf.* Section 4.4.2.3). Further, version two and version three of the payment value evaluator contracts include extended functionality in terms of the reward allocation and the reward distribution.

5.2.3.1 V1ReferralPaymentValueUpgradable

To begin with, the implementation of the V1ReferralPaymentValueUpgradable contracts is comparable to version two of the referral payment quantity contracts. Instead of storing a quantity threshold value to evaluate the number of referral payments done, it saves the `paymentsValueThreshold` value on the contract.(*cf.* Figure 5.6). This value is used to evaluate the referral process not only within this version of the contract but also in versions two and three, which are covered in the upcoming sections.

The `paymentsValueThreshold` is utilized in the `registerReferralPayment` function to evaluate the referral process, as can be observed in Listing C.6. Therein the `paymentsValue` variable of the ReferralProcess struct is compared to the `paymentsValueThreshold`.

Apart from that, the contract's implementation follows the code of version two of the referral payment quantity contracts, as outlined previously (*cf.* Section 5.2.2.2).

5.2.3.2 V2ReferralPaymentValueUpgradable

Next, the UML class diagram in Figure 5.7 demonstrates how version two of the referral payment value contracts differs from version one. Version two of the contract implements

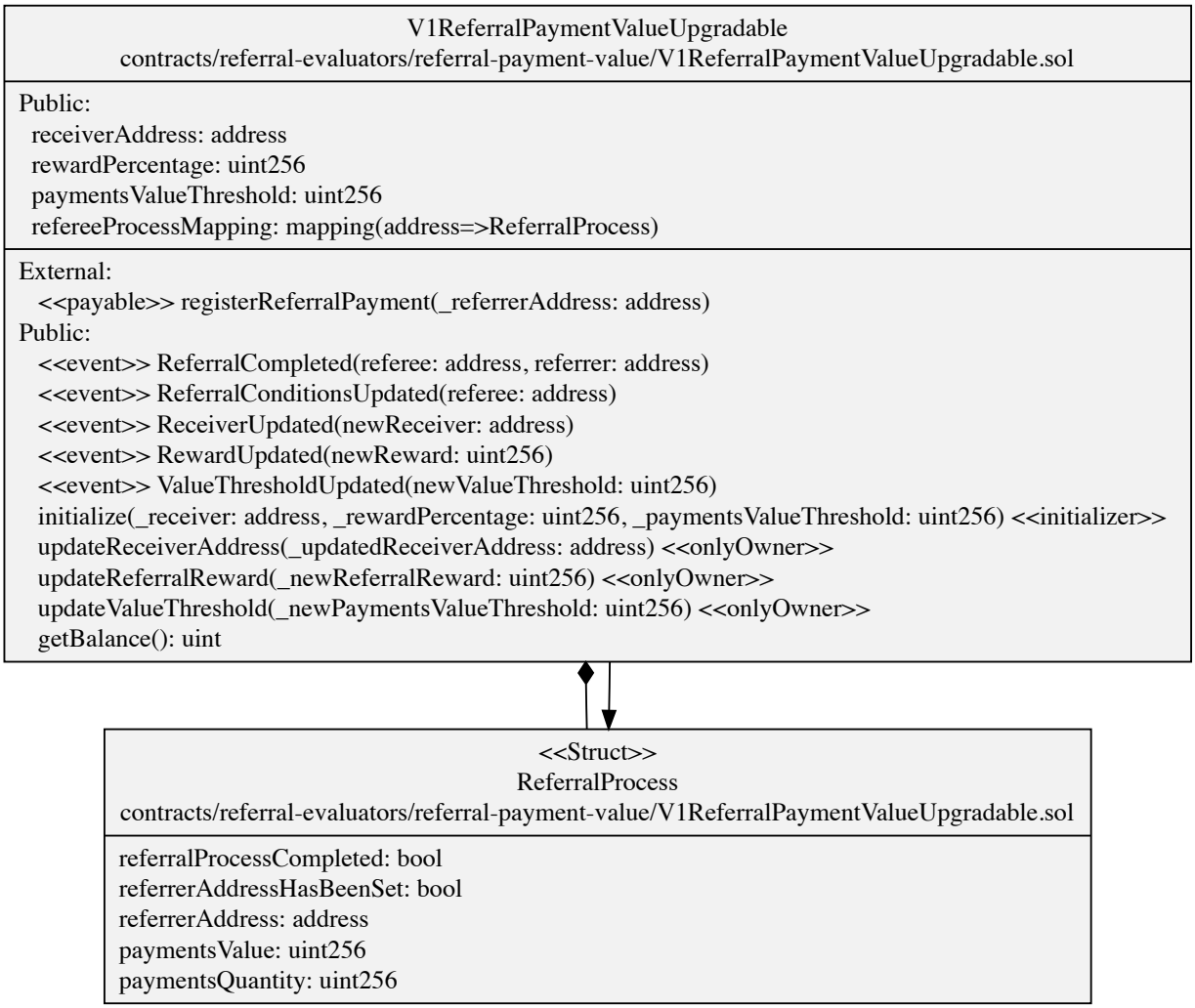


Figure 5.6: UML Class Diagram of the V1ReferralPaymentValueUpgradable Contract

Figure 5.7: UML Class Diagram of the V2ReferralPaymentValueUpgradable Contract

an additional feature related to the reward distribution. Therefore, there are two main differences in the implementation of this version.

The V2ReferralPaymentValueUpgradable contract stores an additional `mapping` named `claimableRewardMapping`, as it is illustrated in Figure 5.7. This mapping assigns a positive numeric `uint256` value to an `address`. The value saved for each `address` in this `mapping` represents the reward that is claimable on the contract. Claimable rewards can be collected or claimed by the users with the respective `addresses` that are assigned to the value in the `claimableRewardMapping`.

```
contract V2ReferralPaymentValueUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    function claimRewards() public {
        uint256 rewards = claimableRewardMapping[msg.sender];
        require(rewards > 0, "No rewards to claim");
        claimableRewardMapping[msg.sender] = 0;
        payable(msg.sender).transfer(rewards);
        emit ClaimedRewards(msg.sender, rewards);
    }
    [...]
}
```

Listing 5.8: `claimRewards` Function of V2ReferralPaymentValueUpgradable

Consequently, the contract also does not send the rewards directly to the eligible users in the `registerReferralPayment` method, as implemented in the previous contracts. The V2ReferralPaymentValueUpgradable contract keeps and stores the assets *i.e.,* the amount of reward an `address` is eligible for in the `claimableRewardMapping`. If the `registerReferralPayment` function in Listing C.7 is compared with previous implementations of this functions *e.g.,* in Listing C.6, the difference becomes evident.

Users can check whether they can claim any rewards in the `mapping`. If users are eligible for rewards, they can call the `claimRewards` function of the contract to retrieve their referral rewards (*cf.* Listing 5.6).

5.2.3.3 V3ReferralPaymentValueUpgradable

The features added in version two for the reward distribution are not implemented for version three of the referral payment value contracts. Thus, the reward distribution is handled as it was done before in version one. Nonetheless, the V3ReferralPaymentValueUpgradable contract implements the functionality to adapt the reward allocation. This is achieved by storing a `refereeRewardPercentage` variable on the contract and adapting the `registerReferralPayment` function of the contract (*cf.* Figure 5.7). The `refereeRewardPercentage` is also restricted to be a percentage value.

As it can be observed in Listing C.8, based on the `calculatedTotalReward`, the `rewardPercentage` and the `refereeRewardPercentage`, the rewards for the referrer and the referee are calculated in the `registerReferralPayment` function if a referral process has

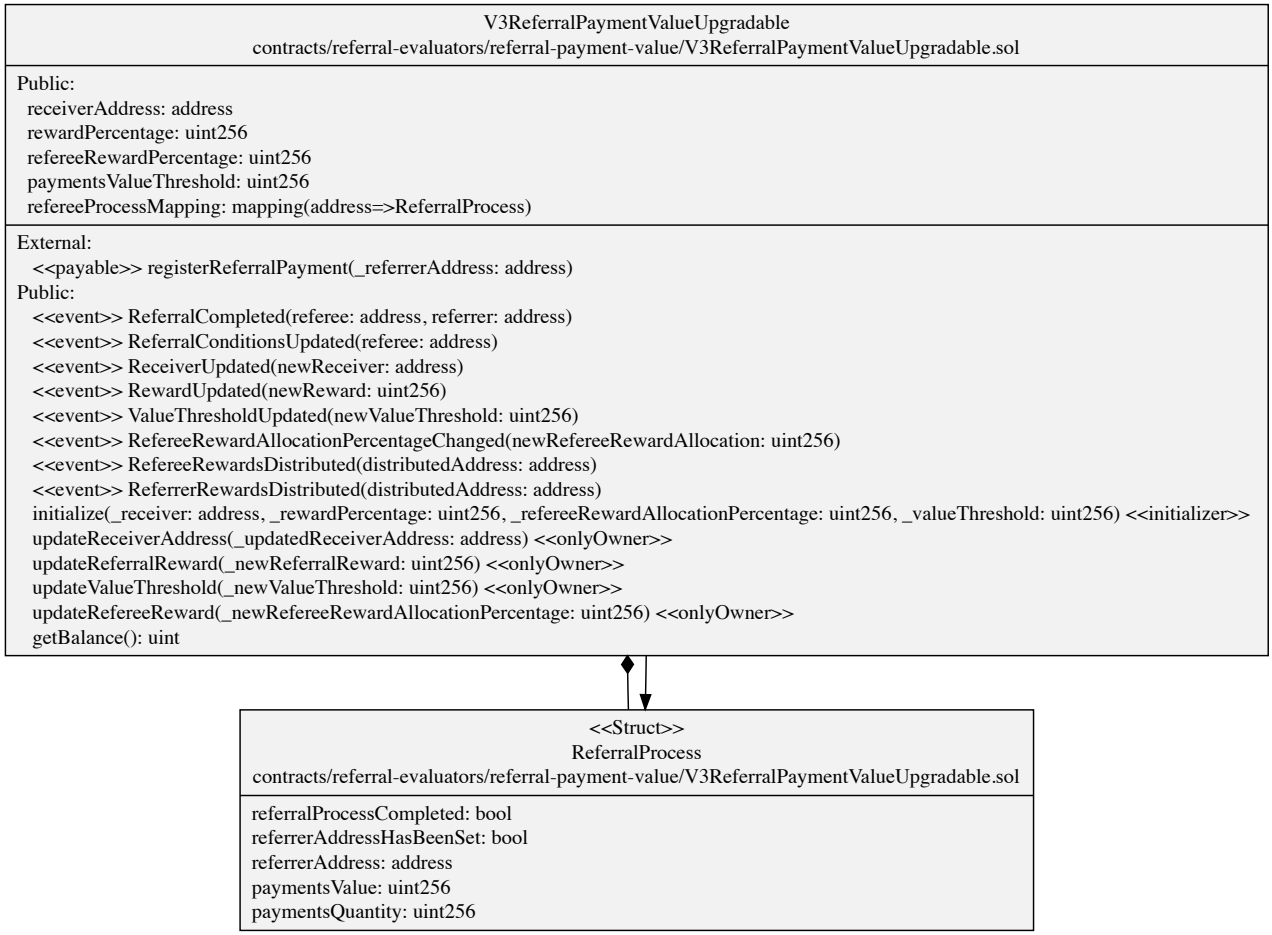


Figure 5.8: UML Class Diagram of the V3ReferralPaymentValueUpgradable Solidity Contract

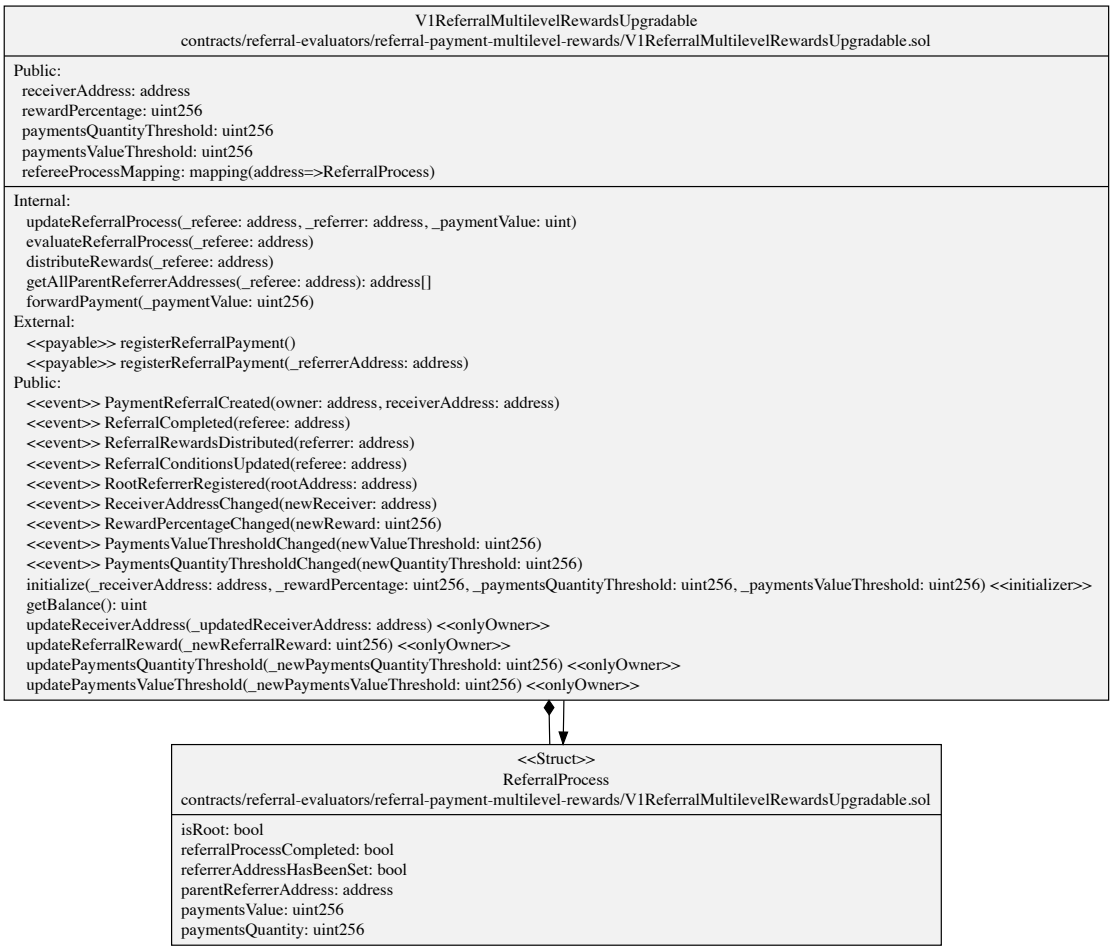


Figure 5.9: UML Class Diagram of the V1ReferralMultilevelRewardsUpgradable Contract

been evaluated as completed. Thereafter, the resulting and potentially two-sided rewards are sent to the referrer as well as the referee.

5.2.4 Referral Payment Multilevel Reward Evaluator Contracts

The referral payment multilevel evaluator contracts include and combine multiple different aspects of the implementation of the referral payment quantity as well as value contracts. Their main characteristic is related to the reward distribution, which can happen on multiple levels as characterized before in Section 4.4.2.4.

Within the previous implementations of solution contracts that have been shown in the sections before, it becomes apparent that the code inside the large functions *e.g.,* for the `registerReferralPayment` function in Listings C.6, C.7 or C.8, is getting increasingly hard to overview and lacks readability. Based on this, and the fact that the payment multilevel contracts combine and introduce new logic and functionality, the code in the upcoming implementation is split into multiple methods again. Thereby, both versions of the payment multilevel contracts incorporate several `internal` functions again to improve the readability of the Solidity code.

5.2.4.1 V1ReferralMultilevelRewardsUpgradable

The UML class diagram of the V1ReferralMultilevelRewardsUpgradable contract in Figure 5.9 portrays that there is a `paymentsQuantityThreshold` value as well as a `paymentsValueThreshold` stored on the contract. Listing 5.9 shows how these two values are used to evaluate a referral process in the `internal evaluateReferralProcess` function. Thereby, the `paymentsQuantityThreshold` is assessed against the `paymentsQuantity` and the `paymentsValueThreshold` against the `paymentsValue` values that are both stored in the `ReferralProcess struct`.

```
1  contract V1ReferralMultilevelRewardsUpgradable is Initializable,
       ↪ OwnableUpgradeable {        [...]
2      function evaluateReferralProcess(address _referee) internal {
3          ReferralProcess storage refereeProcess = refereeProcessMapping[
4              _referee
5          ];
6          // require referrer address has been set
7          require(
8              refereeProcess.referrerAddressHasBeenSet,
9              "Referrer Address has not been set for this referee"
10         );
11         // check if thresholds for payments value and quantity are
               ↪ surpassed
12         if (
13             refereeProcess.paymentsValue > paymentsValueThreshold &&
14             refereeProcess.paymentsQuantity > paymentsQuantityThreshold
15         ) {
16             distributeRewards(_referee);
17             // referral process is completed
18             refereeProcess.referralProcessCompleted = true;
19             emit ReferralCompleted(_referee);
20         }
21     }
22     [...]
23 }
```

Listing 5.9: `evaluateReferralProcess` Function of V1ReferralMultilevelRewardsUpgradable

On closer inspection of the UML class diagram, it also shows that the referral payment multilevel reward contract implements two functions named `registerReferralPayment`. The difference between these functions is that one requires a `_referrerAddress` argument of type `address`, and the other method has no function parameters (*cf.* Figure 5.9). This Solidity concept, called function overloading [128], allows defining multiple functions with the same name but different parameter types within a single contract. Thereon, the appropriate function version is called based on the provided input arguments. The function overloading is needed since, as discussed in Section 4.4.2.4, the multilevel rewards contracts design introduces the restriction that only customers *i.e.,* registered users, can be used as valid referrers. Hence, to become a registered user or customer, there must be a possibility to execute payments to the contract without specifying a referrer `address`. This is handled by the `registerReferralPayment` function overloading. Thus, the first or empty `registerReferralPayment` function with no parameters as it is shown in Listing C.9 registers the sender of the referral payment transaction in the `refereeP-`

rocessMapping as a root referrer. In addition, this function handles the case when the sender has already been registered as a referee. Thereon, it just updates and evaluates the referral process values in the ReferralProcess struct and forwards the payment to the receiver. Hence, already registered referees can also send transactions without a referrer address. Their corresponding referrer address has already been set within their first transaction call of the registerReferralPayment that includes the _referrerAddress parameter shown in Listing C.10.

```
contract V1ReferralMultilevelRewardsUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    function distributeRewards(address _referee) internal {
        ReferralProcess storage completedProcess = refereeProcessMapping[
            _referee
        ];
        // calculate the reward for the referrer
        uint256 calculatedReferrerReward = (completedProcess.paymentsValue
            ↪ /
            100) * rewardPercentage;
        // get all address that are eligible for rewards
        require(
            address(this).balance >= calculatedReferrerReward,
            "Contract has not enough funds to pay rewards"
        );
        // get all eligible referral addresses
        address payable[]
            memory rewardAddresses = getAllParentReferrerAddresses(_referee
                ↪ );

        // calculate reward per referrer
        uint256 numberOfRewardAddresses = rewardAddresses.length;

        uint256 rewardProportion = calculatedReferrerReward /
            numberOfRewardAddresses;

        uint256 i = 0;
        // distribute rewards to all referrers
        while (i < numberOfRewardAddresses) {
            rewardAddresses[i].transfer(rewardProportion);
            emit ReferralRewardsDistributed(rewardAddresses[i]);
            i++;
        }
    }
    [...]
}
```

Listing 5.10: distributeRewards Function of V1ReferralMultilevelRewardsUpgradable

After all, to participate and become a valid referrer, users must either execute the empty registerReferralPayment function outlined in Listing C.9 to become a root referrer or call the regular registerReferralPayment with a valid referrer address and become a referee as it is depicted in Listing C.10.

Furthermore, the two registerReferralPayment functions illustrated in Listings C.9 and C.10 call the evaluateReferralProcess function (*cf.* Listing 5.9) which in turn

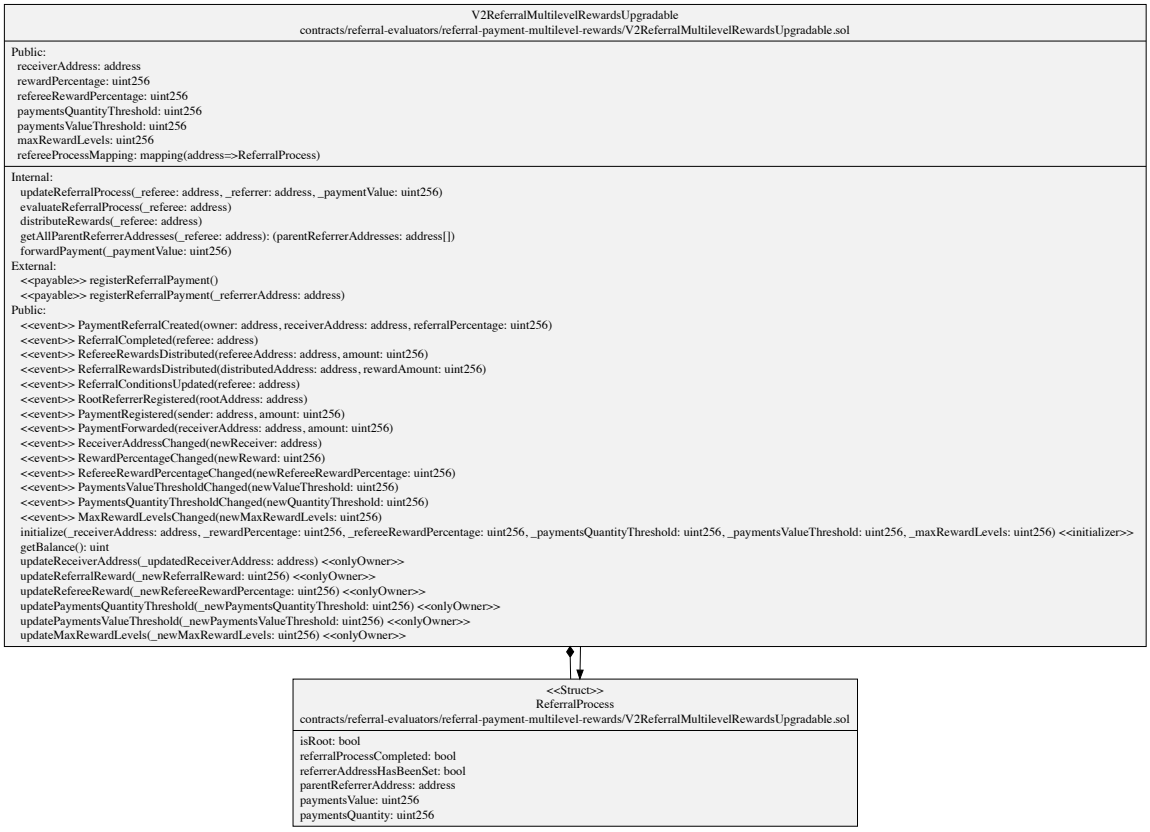


Figure 5.10: UML Class Diagram of the V2ReferralMultilevelRewardsUpgradable Contract

executes the `distributeRewards` method if a referral process has been completed. The `distributeRewards` method can be inspected in Listing 5.10. This `internal` function implements the main functionality of the referral payment multilevel rewards contracts. It calculates the total rewards and determines all the eligible `rewardAddresses`. The `rewardAddresses` are defined by the `internal` `getAllParentReferrerAddresses` function that is delineated in Listing C.11.

The `getAllParentReferrerAddresses` method retrieves the `parentReferrerAddress` for the referee `address` that has completed the process. The `parentReferrerAddress` that is additionally stored in the `ReferralProcess struct` together with a flag value `isRoot` that indicates whether the current user is a root referrer or not. Thereon, the `getAllParentReferrerAddresses` method checks if the parent *i.e.,* `parentReferrerAddress` of the referee is a referee as well or if it is a root referrer. In this way, the function determines all included referees and referrer `addresses` until a root referrer is found.

Eventually, the determined `rewardAddresses` are returned to the `distributeRewards` function, which then splits the rewards equally and sends the reward proportions to all eligible users (*cf.* Listing 5.10)

5.2.4.2 V2ReferralMultilevelRewardsUpgradable

Version two of the referral payment multilevel reward evaluator contracts incorporates two additional features. Therefore the contract stores two additional values, as shown in Figure 5.10.

```
contract V2ReferralMultilevelRewardsUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    function distributeRewards(address _referee) internal {
        ReferralProcess storage refereeCompletedProcess =
            ↪ refereeProcessMapping[
            _referee
        ];

        // calculate the total reward based on the referee payment value/
            ↪ volume
        uint256 calculatedTotalReward = (refereeCompletedProcess.
            ↪ paymentsValue *
            rewardPercentage) / 100;
        require(
            address(this).balance >= calculatedTotalReward,
            "Contract has not enough funds to pay rewards"
        );

        // calculate and distribute referee rewards
        uint256 refereeReward = (calculatedTotalReward *
            refereeRewardPercentage) / 100;
        payable(_referee).transfer(refereeReward);
        emit ReferralRewardsDistributed(_referee, refereeReward);

        // calculate remaining referrer rewards
        uint256 referrerReward = calculatedTotalReward - refereeReward;
        // get all eligible referral addresses
        address payable[]
            memory rewardAddresses = getAllParentReferrerAddresses(_referee
                ↪ );

        // calculate reward per referrer in reward chain
        uint256 numberOfRewardAddresses = rewardAddresses.length;
        uint256 referrerRewardProportion = referrerReward /
            numberOfRewardAddresses;

        // distribute rewards to all eligible referrers
        for (uint256 i = 0; i < numberOfRewardAddresses; i++) {
            rewardAddresses[i].transfer(referrerRewardProportion);
            emit ReferralRewardsDistributed(
                rewardAddresses[i],
                referrerRewardProportion
            );
        }
    }
    [...]
}
```

Listing 5.11: `distributeRewards` Function of V2ReferralMultilevelRewardsUpgradable

The `refereeRewardPercentage` value reintroduces the possibility to define and distribute two-sided rewards, similar to what has been discussed in the implementation of the V3ReferralPaymentValueUpgradable contract in Section 5.2.3.3. Listing 5.11 shows how the amount of the reward that is sent to the referee upon completion is calculated in the `distributeRewards` function.

```
contract V2ReferralMultilevelRewardsUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    function getAllParentReferrerAddresses(
        address _referee
    ) internal view returns (address payable[] memory
        ↪ parentReferrerAddresses) {
        uint256 length;
        address currentRefereeAddress = _referee;

        // loop until get to root address OR maxRewardLevels is reached
        while (
            refereeProcessMapping[currentRefereeAddress].isRoot != true &&
            length < maxRewardLevels
        ) {
            length++;
            currentRefereeAddress = refereeProcessMapping[
                ↪ currentRefereeAddress]
                .parentReferrerAddress;
        }

        parentReferrerAddresses = new address payable[](length);

        currentRefereeAddress = _referee;
        for (uint256 i = 0; i < length; i++) {
            parentReferrerAddresses[i] = refereeProcessMapping[
                currentRefereeAddress
            ].parentReferrerAddress;
            currentRefereeAddress = parentReferrerAddresses[i];
        }

        return parentReferrerAddresses;
    }
    [...]
}
```

Listing 5.12: `getAllParentReferrerAddresses` Function of V2ReferralMultilevelRewardsUpgradable

Moreover, the V2ReferralMultilevelRewardsUpgradable contract implements a `maxRewardLevels` value. As it has been illustrated in the design of this contract (*cf.* Section 4.4.2.4), this value is used to limit the number of referrers or parent referrers rewards are distributed to. Thereby, the `maxRewardLevels` is applied in the `getAllParentReferrerAddresses` function as it is displayed in Listing 5.12.

Apart from these two additional features, there are no other major differences in the implementation of versions one and two of the multilevel rewards contracts.

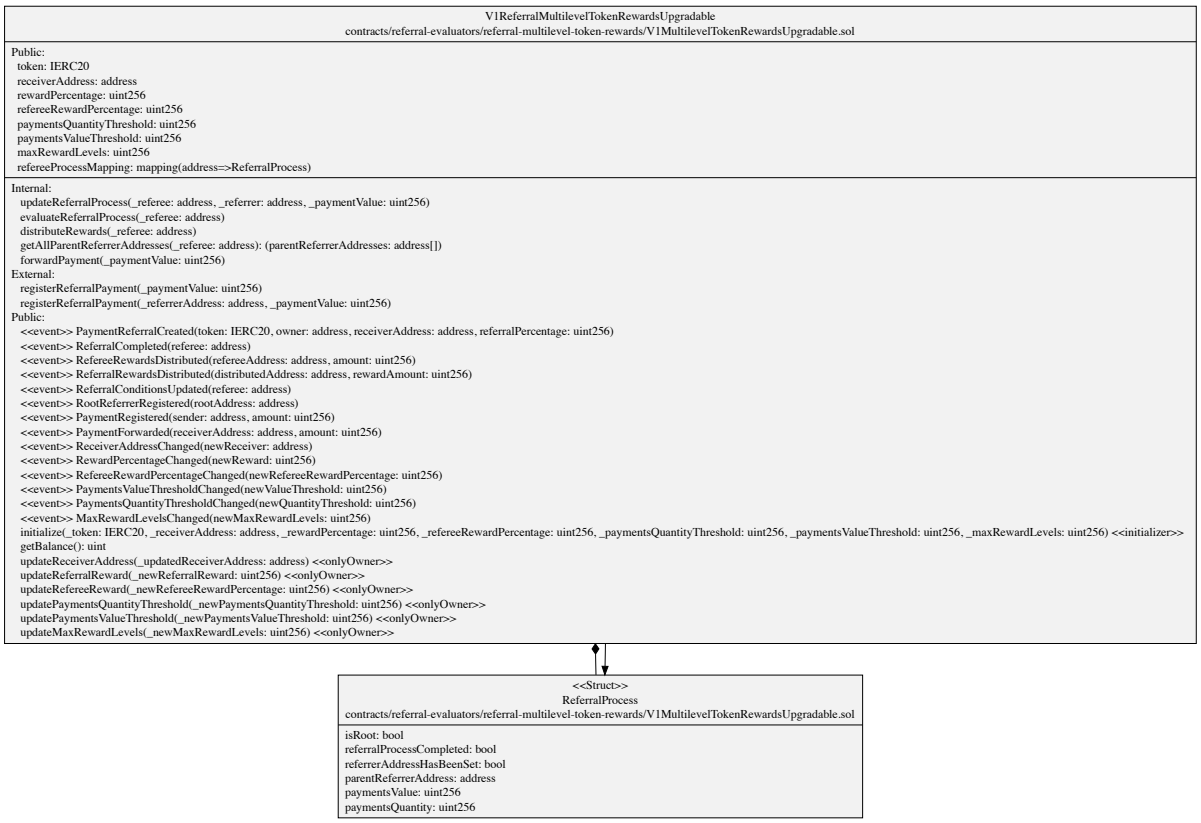


Figure 5.11: UML Class Diagram of the V1ReferralMultilevelTokenRewardsUpgradable Contract

5.2.5 Referral Payment Multilevel Token Reward Evaluator Contracts

The V1ReferralMultilevelTokenRewardsUpgradable contract is the last Deferral solution smart contract that has been implemented. The implementation of this token reward contract follows the V2ReferralMultilevelRewardsUpgradable contract. However, this token reward contract is adapted so the referral process, including accepting payments and distributing rewards, works with a defined ERC20 token instead of native assets.

5.2.5.1 V1ReferralMultilevelTokenRewardsUpgradable

Figure 5.11 shows that compared to version two of the multilevel rewards contract (*cf.* Figure 5.10), there are not many differences in the UML class diagrams. This contract additionally stores a `token` value on the contract. The `token` value defines the ERC20 token that is set and accepted as currency for referral payment transactions. Hence, users can only execute payments sending assets of this ERC20 `token`. Moreover, the UML class diagram shows that there is no update method for this `token` value compared to all other variables stored on the contract (*cf.* Section 4.4.2.5).

Furthermore, Figure 5.11 reveals that multiple functions of the token contract had to be adjusted *e.g.,* by including a `_paymentValue` parameter that has to be passed to indicate the amount of ERC20 tokens that is sent within the referral payment transaction. In contrast to the native assets, the information about the amount of ERC20 tokens sent within

a transaction is not included in the default transaction data. Thus, the implementation of functions like the two `registerReferralPayment` or the `forwardPayment` method had to be slightly adjusted.

The small code changes can also be seen in Listing C.12 that outlines the `registerReferralPayment` function implementation. Therein, it can be seen how the amount of token that is specified in the `_paymentValue` argument of the function called is transferred from the sender to the contract. The V1ReferralMultilevelTokenRewardsUpgradable contract then transfers a certain amount of these tokens further to the receiver. The remaining tokens are kept on the contract until the referral process has been completed. Thereafter the `distributeRewards` function, illustrated in Listing C.13, transfers the reward tokens to all eligible referrers and or referees.

Using an ERC20 token as the referral system currency is the only functionality added to the solution contract designs, which cannot be reverted or neutralized by setting or updating the values on the contract. The two-sided reward distribution can be set or unset by adjusting the `refereeRewardPercentage` value on the contract. Thus, by setting this value to 0%, the reward allocation remains one-sided for the referrer only. The same applies to the multilevel reward distribution. By setting the `maxRewardLevels` value to equal 1, rewards are only paid out on one level. However, the token reward contract cannot work with native cryptocurrency assets and all other contracts are incompatible with using an ERC20 token for payments.

For further code or details about the implementation of the multilevel token rewards contract or any other solution contract that has been covered in the last few sections, please refer to the Deferral repository on GitHub [132].

5.3 Tests and Testing

All the presented solution smart contracts have been tested with the help of Hardhat and its built-in local Hardhat network [62]. It is possible to run the tests on a different network. Nonetheless, all tests have been outlined to be run and executed on the local Hardhat network since the configured accounts or users in this network have enough funds to test the referral system functionality of the different contracts.

```
1    [...]
2    "test": "npx hardhat test --network hardhat",
3    "test-transmitter-v1": "npx hardhat test test/referral-payment-
        ↪ transmitter/V1ReferralPaymentTransmitter.test.ts",
4    "test-transmitter-v2": "npx hardhat test test/referral-payment-
        ↪ transmitter/V2ReferralPaymentTransmitter.test.ts",
5    "test-transmitter-v3": "npx hardhat test test/referral-payment-
        ↪ transmitter/V3ReferralPaymentTransmitter.test.ts",
6    "test-transmitter-upgrades": "npx hardhat test test/referral-payment-
        ↪ transmitter/UpgradableReferralPaymentTransmitters.test.ts",
7    "test-quantity-v1": "npx hardhat test test/referral-payment-quantity/
        ↪ V1ReferralPaymentQuantity.test.ts",
8    "test-quantity-v2": "npx hardhat test test/referral-payment-quantity/
        ↪ V2ReferralPaymentQuantity.test.ts",
```

```
    "test-value-v1": "npx hardhat test test/referral-payment-value/
        ↪ V1ReferralPaymentValue.test.ts",
    "test-value-v2": "npx hardhat test test/referral-payment-value/
        ↪ V2ReferralPaymentValue.test.ts",
    "test-value-v3": "npx hardhat test test/referral-payment-value/
        ↪ V3ReferralPaymentValue.test.ts",
    "test-multilevel-v1": "npx hardhat test test/referral-payment-
        ↪ multilevel-rewards/V1MultilevelRewardReferral.test.ts",
    "test-multilevel-v2": "npx hardhat test test/referral-payment-
        ↪ multilevel-rewards/V2MultilevelRewardReferral.test.ts",
    "test-token-multilevel-v1": "npx hardhat test test/referral-multilevel-
        ↪ token-rewards/V1MultilevelTokenRewardReferral.test.ts",
    [...]
}
```

Listing 5.13: Excerpt of the Test Scripts in the `package.json` File

For every Solidity contract, a separate `.test` file that includes all test cases for a particular contract has been implemented. Overall, a total of 188 test cases have been written in 12 separate test files for the 11 different solution smart contracts. Since a lot of functionality is shared among the different contracts, the test files often share lines of duplicated code or test cases. Nevertheless, it was important to test each solution contract individually. Thereby, if the implementation of any of the contracts was adapted or modified, it could easily be checked if the referral system still operated as intended with the help of the different test cases. The bigger the logic and features included in a contract, the more code and test cases in the corresponding test file had to be implemented. Information about the test coverage is discussed in the upcoming Chapter 6.

```
    [...]
    it(`${CONTRACT_NAME} should update the referral process data correctly
        ↪ during the uncompleted referral process`, async () => {
      const { referrer, referee, proxyContract } = await loadFixture(
        processTestingFixture
      );
      const initialBalance: BigNumber = await referrer.getBalance();
      // execute n payments in order for the referral process to NOT be
          ↪ completed
      await executeReferralPayment({
        executions: ptPaymentsQuantityThreshold.toNumber(),
        referee,
        referrer,
        proxyContract,
        paymentValue: ptPaymentAmount,
      });
      const referralProcessMapping = await proxyContract.
          ↪ refereeProcessMapping(
        referee.address
      );
      // assert data is updated correctly
      expect(referralProcessMapping.referralProcessCompleted).to.equal(
          ↪ false);
      expect(referralProcessMapping.referrerAddressHasBeenSet).to.equal(
          ↪ true);
      expect(referralProcessMapping.referrerAddress).to.equal(referrer.
          ↪ address);
```

```
22          expect(referralProcessMapping.paymentsValue).to.equal(
23            ptPaymentAmount.mul(ptPaymentsQuantityThreshold)
24          );
25          expect(referralProcessMapping.paymentsQuantity).to.equal(
26            ptPaymentsQuantityThreshold
27          );
28          // assert reward has not been paid out
29          const afterReferrerBalance: BigNumber = await referrer.getBalance();
30          const contractBalance: BigNumber = await proxyContract.getBalance();
31
32          // assert balances are correct afterwards
33          expect(afterReferrerBalance).to.be.closeTo(
34            initialBalance,
35            TEST_PRECISION_DELTA
36          );
37          expect(contractBalance).to.be.closeTo(
38            ptPaymentAmount
39              .mul(ptRewardPercentage)
40              .div(100)
41              .mul(ptPaymentsQuantityThreshold),
42            TEST_PRECISION_DELTA
43          );
44      });
45      [...]
46  }
```

Listing 5.14: Example Test Case of the `V1ReferralPaymentQuantity.test.ts` Test File

An exemplary test case is shown in Listing 5.14. The displayed case tests if the `ReferralProcess` data that is stored for every referee is updated correctly during the execution of referral payment transactions where the referral process has not been completed yet for the V1ReferralPaymentQuantityUpgradable Solidity contract.

For every referral contract *i.e.,* referral system, test cases covering the contract deployment, updating or changing the values stored on the contract, the functionality of the different methods, including function modifiers, and all the logic and processes of the respective referral process have been implemented. All the Hardhat test cases can be executed by using the Command Line Interface (CLI). The `npx hardhat test -network hardhat` command runs all test cases in the local Hardhat network. This command also shows how the tests can be run with different `-network` flags on any other network that has been defined in the `hardhat.config.ts` setup file. A single test file can be run by using the `npx hardhat test test/test-file.test.ts` command. Lastly, for a better and quicker developer experience, several yarn [151] scripts have been defined to run the different test files individually. Listing 5.13 shows an excerpt of the `package.json` file with the different `yarn test` scripts.

For more information about the test setup or the general Hardhat setup of the Deferral project, the respective `README.md` files in the Deferral repository [132] or the official Hardhat documentation [61] can be consulted.

5.4 Scripts

Alongside the referral smart contracts, several scripts have been implemented as part of the solution of this thesis. All the implemented scripts are included in the Deferral [132] and the Deferral visualizations [133] GitHub repositories.

The scripts have been implemented to either deploy the contracts, evaluate the performance of the referral systems, or visualize the results of the evaluations. The deployment and evaluation scripts have been implemented in TypeScript [135] and can be run via the CLI using Hardhat [61]. The visualization scripts are written in Python [109]. In the following sections, the different types of scripts are quickly introduced.

5.4.1 Deployment Scripts

The deployment scripts' main purpose is to test if the Solidity smart contracts could be used and deployed to the different networks configured in the Hardhat environment. Hence, besides the ability to deploy a contract via the CLI, they also support the testing of the deployment of the contracts. Again, for every solution contract, there is a separate deployment script. Thus, there are 11 deployment scripts in total.

The deployment scripts can be executed with Hardhat via the CLI. The `hardhat run scripts/deployment/script-file-name.ts` command executes a specific deployment script. As has been done before for the tests, multiple yarn scripts have been added to the `package.json` file to facilitate the deployment of the different contracts.


```
[
    {
        "id": 0,
        "date": "2023-04-27T14:12:36.861Z",
        "contract": "V1ReferralMultilevelRewardsUpgradable",
        "contractAddress": "0x0549a3532Bd73C3F1b48c61beD23682095B4f502",
        "signer": "0x43b3E4bc4443cb32242b283DF4f2aF9AAe77C4DB",
        "gasUsed": "728329",
        "effectiveGasPrice": "1679631005",
        "cost": "1223323970240645",
        "durationInMs": 741.1912080002949
    },
    {
        "id": 1,
        "date": "2023-04-27T14:14:35.915Z",
        "contract": "V1ReferralMultilevelRewardsUpgradable",
        "contractAddress": "0x0549a3532Bd73C3F1b48c61beD23682095B4f502",
        "signer": "0x43b3E4bc4443cb32242b283DF4f2aF9AAe77C4DB",
        "gasUsed": "728329",
        "effectiveGasPrice": "1679631005",
        "cost": "1223323970240645",
        "durationInMs": 692.9108750000596
    }
]
```


Listing 5.15: V1ReferralMultilevelRewardsUpgradable Deplyoment Log File

One additional feature that is included in all the deployment, as well as evaluation scripts, is logging. For every contract that is deployed via its deployment script, certain metrics are recorded and stored in a `.json` file that is automatically created and stored inside the `logs/deployments/` directory on the local machine. The log files are further arranged by the network a contract has been deployed to.

Listing 5.15 shows the log file for the deployments of version one of the multilevel rewards contract after the corresponding deployment script has been executed twice on the Hardhat network. It shows the different metrics that are recorded for every contract deployment. Among others, the metrics include the `date, contractAddress, gasUsed, durationInMs` and other information about the contract deployment. Eventually, the resulting deployment log file of the V1ReferralMultilevelRewardsUpgradable contract deployment is located in the `referral-payment-multilevel-rewards/Hardhat-Local_31337/` folder inside the `logs/deployments/` directory as mentioned before. If this contract is deployed on the same network again, this file or list is extended with the newly recorded data and information of this deployment.

5.4.2 Evaluation Scripts

The evaluation scripts are the most important scripts. They are responsible for evaluating the solution smart contracts and generating and collecting data about their performance which can then be analyzed and visualized. Hence, the evaluation scripts generate the results data that are used as a foundation for the evaluation and discussions in Chapter 6. However, the scripts are implemented to be easily executed and can generate new results data quickly. In total, there are 10 different evaluation scripts.

In every evaluation script, the corresponding Solidity smart contract is deployed to the Hardhat network. In addition, the gas costs and fiat prices for different evaluation blockchains are fetched and retrieved within every evaluation script. The different evaluation blockchains and how these values are used are covered in the next chapter (*cf.* Chapter 6). Thereon, a certain amount of user or Hardhat accounts are initialized and set up. These accounts represent the users of the referral system. A few of these accounts are used to represent the deployer of the contract, the receiver of the payments, or, in some cases, the initial root referrer. All the remaining accounts represent users *i.e.,* referees and referrers of the referral system. Thus, in an example where the evaluation is done with ten users, two or three accounts, depending on the evaluated solution contract, are used to represent the contract deployer, the payment receiver, and, if required, the root referrer. Consequently, in this evaluation run, seven or eight users participate as referees and complete the referral process.

The number of users is defined by the `NUMBER_OF_EVALUATION_ACCOUNTS` env variable that can be stored in a `.env` file. More instructions about the env variables and the `.env` file can be found in the `README.md` file in the Deferral repository [132].

All the evaluation scripts can also be executed with Hardhat. To execute a contract-specific evaluation, the `hardhat run scripts/evaluation/evaluation-file-name.ts` command can be run via the CLI. Within the CLI command, the number of users can also

be adapted by adding `USE_EVALUATION_ACCOUNTS=true` and `NUMBER_OF_EVALUATION_ACCOUNTS=x` statements to the Hardhat command, where `x` represents the number of total users for the evaluation run.

```
[...]
 // number of txs per user to complete the referral process
  const txsPerUser = QUANTITY_THRESHOLD.toNumber() + 1;

  // -1 since we need a referrer for every referee
  const loopIterations = numberOfUsers - 1;

  // execute transactions for signers
  for (let i = 0; i < loopIterations; i++) {
    const refereeUser = users[i];

    // referrer user (addressed used as referrer address)
    const referrerUser = users[i + 1];
    // execute required amount of txs per user to complete referral
    for (let j = 0; j < txsPerUser; j++) {
      // referee user (executes the txs)

      const txStartTime = performance.now();

      // execute the referral payment transactions / complete referral
          ↪ process
      const referralPaymentTx = await proxyContract
        .connect(refereeUser)
        .registerReferralPayment(referrerUser.address, {
          value: PAYMENT_AMOUNT,
        });
      const txEndTime = performance.now();

      // get referral process status
      const mapping = await proxyContract.refereeProcessMapping(
        refereeUser.address
      );
      const referralCompleted: boolean = mapping.referralProcessCompleted;

      // calculate tx evaluation data, including gas costs and fiat prices
      const txEvaluationData: TransactionEvaluationValuesType =
        await getTxEvaluationData(
          txStartTime,
          txEndTime,
          referralPaymentTx,
          bnGasPricesInWei,
          fiatPrices
        );
[...]
```

Listing 5.16: Excerpt of the V1ReferralPaymentQuantityUpgradable Evaluation Script

Every evaluation follows the same procedure adapted to the implementation of the corresponding referral contract. An excerpt of the evaluation script for the V1ReferralPaymentQuantityUpgradable contract is illustrated in Listing 5.16. The evaluation script completes a referral process for all users based on the referral conditions defined by the arguments and values used during the contract deployment.

Hence, if it takes three transactions to complete a referral process, since *e.g.,* the payment quantity threshold in the referral conditions is set to two, three transactions are executed for every user. Otherwise, if only one transaction is required to complete the referral process for the evaluation of a simple payment transmitter referral contract, only one transaction is executed. Thus, the number of transactions executed per user depends on the number of referral payment transactions that are required to complete the referral process (*cf.* `txsPerUser` value in Listing 5.16). The contract's deployment arguments *i.e.,* the referral conditions, can be set in the code itself and are implemented so they can easily be adjusted.

As outlined before for the deployment scripts, the evaluation scripts create and store local log files that collect data about the different evaluation processes and transactions. The only difference is that the evaluation log files are stored and located in the `logs/evaluations/` directory. More details and information about what exact data and evaluation metrics are collected and how they are calculated are discussed in the upcoming Chapter 6.

5.4.3 Visualization Scripts

The visualization scripts help to visualize the data generated from the contract evaluation and the evaluation scripts. The scripts and their complete implementation can be found in the Deferral Visualizations submodule repository [133]. The data was arranged within these scripts and eventually visualized into several charts using Plotly [104]. In total, there are four Python visualization scripts.

First, there is the `transaction_evaluations_script` which loads and reads the transaction evaluation data of all results and visualizes the recorded and executed referral payment transaction. Next, the `evaluation_runs_evaluation_script` evaluates the result data of the evaluation runs made with multiple numbers of users. Further, there is a `overall_evaluation_script` script that produces visualizations useful for the overall evaluation and analysis of the results.

Last, there is the `historic_price_evaluation_script`, which not only visualizes but also analyses and fetches historic gas price and cryptocurrency price data via the Owloracle API [98]. More details about the evaluation done in the visualization scripts can be found directly in the code or in the `README.md` file of the visualization repository [133].

5.4.4 Shell Scripts

Lastly, a few different shell scripts have been implemented. These scripts are not relevant to the final solution of this thesis. However, they improve the developer experience for deploying and evaluating the solution contracts. Broadly, the `deploy.sh` script executes all the deploy statements at once so all the contracts can be deployed by running one command in the CLI. Moreover, especially the `evaluate.sh` shell script makes evaluating the different contracts more straightforward. This script offers the possibility to adjust the number of users in an array with values representing the number of users that should

be used in a complete evaluation run. In a complete evaluation run, all evaluation scripts are executed once with the corresponding number of test users. For the visualizations, a `generate_visualization_results.sh` script has been implemented, which runs all three Python visualization scripts at the same time.

Chapter 6

Evaluation and Discussion

Within this chapter, the Deferral solution, including the various referral smart contracts, is evaluated. In the process, the results of the evaluation are discussed before they are assessed against the overall goal and solution requirements of this thesis. Eventually, the results and final solution of this thesis are reviewed again in a broader scope within the context of decentralized referral systems.

To begin with, the security of the referral smart contracts is evaluated by having a look at the resulting test metrics. Thereafter, the primary focus lies on the analysis and comparison of the data generated by the evaluation scripts (*cf.* Section 5.4.2). Based on this evaluation data, the different smart contract solutions are benchmarked with regard to costs and performance. The contracts are evaluated first versus the different versions within the same solution design and second across all different solution designs and contracts that have been introduced in Chapter 4.

6.1 Test Coverage and Security

The security of the Deferral solution and its different smart contracts is mainly ensured by the Hardhat tests. As it has been discussed in Section 5.3, a number of test cases have been implemented to assess the functionality and security of the contracts. Thereby the test results that are outlined in Table 6.1 show the overall coverage for the corresponding Solidity smart contract implementations. The code coverage is split up into measures for statements, branches, functions, and lines coverage.

In Table 6.1, it can be observed that, aside from branch coverage, all other coverage metrics achieve 100%. In summary, this demonstrates a well-tested evaluation of the security of smart contracts. Nonetheless, it is important to note that the test cases primarily focus on examining the basic functionalities of smart contracts, including completion and cycle through the referral process. Therein the execution and evaluation of referral payments, as well as the accurate reward distribution, is prioritized. The test cases are mainly designed to assess the feasibility of the solution contracts within the context of this thesis, assuming a happy path for the referral process in general.

Referral Contract	Test Coverage			
	% Stmts	% Branch	% Funcs	% Lines
referral-multilevel-token-rewards/	100	90.74	100	100
V1MultilevelTokenRewardsUpgradable	100	90.74	100	100
referral-payment-multilevel-rewards/	100	90	100	100
V1ReferralMultilevelRewardsUpgradable	100	86.84	100	100
V2ReferralMultilevelRewardsUpgradable	100	92.31	100	100
referral-payment-quantity/	100	86	100	100
V1ReferralPaymentQuantityUpgradable	100	84.62	100	100
V2ReferralPaymentQuantityUpgradable	100	87.5	100	100
referral-payment-transmitter/	100	95	100	100
V1ReferralPaymentTransmitter	100	100	100	100
V2ReferralPaymentTransmitterUpgradable	100	75	100	100
V3ReferralPaymentTransmitterUpgradable	100	95	100	100
referral-payment-value/	100	89.47	100	100
V1ReferralPaymentValueUpgradable	100	87.5	100	100
V2ReferralPaymentValueUpgradable	100	91.67	100	100
V3ReferralPaymentValueUpgradable	100	89.29	100	100
All files	100	90	100	100

Table 6.1: Test Coverage Report for all the Referral Smart Contracts

Although the test coverage results lend support to the feasibility of the Deferral solutions, further testing needs to be done to improve the security of the smart contracts. The test cases do not inherently offer a comprehensive testing of the referral process and all potential edge cases, which is also reflected in the branch coverage that hovers around 90% (*cf.* Table 6.1). Further, to improve the security of the Solidity smart contracts beyond the achieved high test coverage, it would be essential to conduct thorough peer code reviews and perform other verification *e.g.,* in the form of professional security reviews. Eventually, the progressions to a real-world applicable integration of these solutions as a decentralized referral system, and various additional aspects, including end-to-end and real user tests, would be required.

After all, the comprehensive test cases implemented using Hardhat and the high test coverage they achieve endorse the feasibility of the Deferral solution in terms of security while leaving room for future improvements.

6.2 Costs and Performance

The main part of the Deferral solution evaluation deals with the costs and performance of the implemented Solidity smart contracts and their different designs. Therein the emphasis is placed on two aspects. First, the individual solution contracts grouped by their solution designs are discussed and evaluated. Here, the focus lies mainly on the costs concerning the collected gas used values. Still, all the other collected and calculated evaluation measures (*cf.* Section 6.2.1) are either attached in the Appendix or can be

found in the `results` folder in the Deferral repository [132].

Second, the overall results are evaluated across all the different solution contracts. This broader evaluation incorporates external aspects such as gas and fiat prices and the resulting calculated evaluation metrics. More information about the evaluation method, including the detailed data and metrics collected and fetched for the evaluation, is introduced in the upcoming Section 6.2.1.

In general, the data used for the evaluations comes from two main sources. On the one hand, the data collected and generated by the evaluation (*cf.* Section 5.4.2) and visualization (*cf.* Section 5.4.3) scripts are analyzed. The evaluation data and metrics are established in the upcoming section extending on Section 5.4.2. On the other hand, the output of the Hardhat Gas Reporter plugin [32] is considered and analyzed against the results of the evaluation scripts. The plugin integrates with the Hardhat testing framework and collects data during the execution of the implemented test cases. The reported output provides insights into the efficiency of the Deferral smart contracts. In addition, external data for the gas and fiat prices or historic data is fetched from different Application Programming Interfaces *(APIs)*.

6.2.1 Evaluation Method and Data

The evaluation scripts include several configurations and combine different measures and metrics. First of all, data about the gas prices for different EVM-based blockchains are fetched. These blockchains have been predefined to be used for the evaluation process and are all EVM-based. The following list shows the defined EVM-based evaluation blockchains and their native cryptocurrencies:

- **Ethereum**: The native currency of Ethereum is *Ether (ETH)*.
- **Binance Smart Chain**: The native currency of Binance Smart Chain is *Binance Coin (BNB)*.
- **Polygon**: A Layer-2 solution where the native currency is *MATIC (MATIC)*.
- **Arbitrum**: A Layer-2 solution where the native currency is *Ether (ETH) e.g.,* the gas fees are paid in Ether.
- **Optimism**: A Layer-2 solution where the native currency is is *Ether (ETH) e.g.,* the gas fees are paid in Ether.
- **Avalanche**: The native currency of Avalanche is *AVAX (AVAX)*.
- **Goerli**: An Ethereum testnet where the native currency is *Goerli Ether (gETH)*.

In EVM-based evaluation blockchains, gas refers to the unit of measurement required to quantify the computational effort needed to execute transactions (*cf.* Section 2.3.4). The gas used for a transaction is the foundation that, in combination with the gas price, defines

how expensive the execution of that particular transaction on a certain blockchain is. EVM-based blockchains share the same principle of gas and use the EVM to execute smart contracts. Thus, the gas used for specific transactions, operations, and computations undertaken on a Solidity smart contract is the same or similar on two different EVM-based blockchains. Within the context of this thesis, the gas used metric is expected to be identical on all the outlined EVM-based evaluation chains for the generation and evaluation of the results. Nevertheless, the gas costs, which are the product of the gas used and the gas price of a particular chain, can significantly vary across different EVM-based blockchains.

6.2.1.1 Gas Prices

The gas price on a particular blockchain represents the cost of one unit of gas on that blockchain. Factors such as the network occupancy rate or user demand can impact the gas price on a particular blockchain. Gas is usually paid in the native currency of the blockchain. For example, on Ethereum, one gas could cost 10 Ether, and on the Binance Smart Chain, one gas could cost 5 BNB. Several layer-2 solutions chains, including Arbitrum or Optimism, also use Ether to pay gas fees. Other layer-2 solutions, such as Polygon, have their own native currency *e.g.,* MATIC, to pay the gas fees. Furthermore, gas prices are usually lower on layer-2 solution blockchains (*cf.* Section 2.3.6) such as Polygon, Arbitrum, or Optimism.

In general, gas prices are not the same across different blockchains and can heavily fluctuate over time on the same blockchain. Therefore, even if the gas used for a transaction execution stays the same across EVM-based blockchains, the different gas prices imply how much a transaction costs on a certain chain. Consequently, it is important for the evaluation to get the current gas prices of the different evaluation chains. The gas prices are fetched with the help of `Ethers.js` [55]. However, gas prices and the resulting calculated gas costs of transactions can be misleading as they are represented in the corresponding cryptocurrency of the blockchain. Thus, they often do not share a common denominator.

6.2.1.2 Fiat and Cryptocurrency Prices

After the gas prices have been retrieved for the different evaluation chains, the evaluation process continues by fetching the prices for the different native cryptocurrencies. The price or fiat price of native cryptocurrencies of a specific blockchain refers to their value when exchanged for other currencies, such as USD or Euros. Consequently, fiat prices across different blockchains can share the same fiat denominator *e.g.,* the price in USD.

Again cryptocurrency prices can be very volatile due to known factors like supply and demand, the general market situation, investor sentiment, and many other conditions. For retrieving the cryptocurrency prices during the evaluation, the CoinGecko API is used [38].

6.2.1.3 Evaluation Metrics

After the data for the gas prices as well as cryptocurrency prices of all the evaluation chains have been collected, the evaluation continues with the actual assessment of the referral systems *i.e.,* smart contracts. First, the participating users are initialized and set up. The number of users is an independent variable and another important factor for the evaluation. A Deferral solution contract *i.e.,* referral system, may deal with the load and respond differently if there are increasingly more users participating or accessing the system. In this case, the effect of the number of users is evaluated primarily based on cost and duration while giving less consideration to factors such as transaction throughput, latency, or frequency of the different evaluation blockchains.

With each of the different defined numbers of users, an evaluation run is executed. An evaluation run refers to one complete execution of the referral script (*cf.* Section 5.4.2) with a particular number of users that are completing the referral process. As part of an evaluation run, data and metrics are collected twofold.

First, transaction evaluation data is gathered for every executed referral payment transaction and stored in log files for later analysis and evaluation. The following metrics and indicators are stored for every referral payment transaction execution:

- **Transaction Evaluation Metrics:**
  - **UserSignerAddress**: Address of the user executing the referral payment transaction.
  - **UserIteration**: Identifier for the user count.
  - **UserTxIteration**: Identifier for the transaction count of the same user.
  - **DurationInMs**: Duration of the referral payment transaction in milliseconds.
  - **GasUsed**: Amount of gas used for the execution of the referral payment transaction.
  - **GasCost**: Gas cost ($gasUsed * gasPrice$) for every transaction for all evaluation chains based on the respective gas price of a chain. Depending on the execution date and time, these values can fluctuate.
  - **FiatCost**: Fiat cost ($gasCost * fiatPrice$) for every transaction for all evaluation chains based on the defined fiat currency and respective fiat price of a chain. Depending on the execution date and time, these values can fluctuate.

Based on these measures and metrics, the referral payment transactions within one complete evaluation run can be analyzed. The following metrics and measurements are collected and calculated for every evaluation run:

- **Evaluation (Run) Metrics:**
  - **ID**: Identifier for the evaluation run.
  - **ContractName**: Name of the evaluated referral smart contract.

- **Network**: Network identifier where the contract was evaluated.
- **Date**: Execution date and timestamp of the evaluation.
- **DurationInMS**: Duration of the whole evaluation run in milliseconds.
- **EtherUnit**: EVM Unit used for the values in the scripts (*cf.* Section 2.3.5).
- **ContractParameters**: Parameter the evaluated referral contract was deployed with.
- **FiatPriceCurrency**: The defined currency for fetching the fiat price values *e.g.,* USD.
- **ChainFiatPrices**: Recorded fiat prices of the defined evaluation blockchains. Depending on the `Date`, these values can fluctuate.
- **ChainGasPrices**: Recorded gas prices of the defined evaluation blockchains. Depending on the `Date`, these values can fluctuate.
- **NumberOfUsers**: Total number of users involved in the evaluation run.
- **Metrics**: Calculated metrics including `avg, min, max, median` and `sum`, of all transaction evaluation metrics recorded, *e.g.,* `durationInMs, gasUsed, gasCost, fiatCost`, etc..., for all executed payment transactions within one particular evaluation run.
- **Data**: All recorded transactions and the collected transaction evaluation data. This data is used to calculate the metrics of an evaluation run

Consequently, if three evaluation runs are done with 10, 20, and 30 users, the evaluation log files include three entries that each entail the evaluation run metrics and measures shown in this list as well as all the transaction evaluation metrics outlined in the list before.

6.2.1.4 Historic Gas and Fiat Price and Cost Data

Furthermore, it is important to consider that components such as the gas or fiat prices used for the calculation of these evaluation metrics, and the result data can heavily fluctuate depending on the date and time of the execution of the evaluation scripts. As a result, to better estimate and assess the results related to the fetched gas and fiat prices, a separate script collects and evaluates historic data of these values (*cf.* Section 5.4.3). Thereby, the gas prices, as well as fiat prices, were retrieved and fetched for a certain period of time.

The historic gas and fiat prices are fetched with the help of the Owloracle API [98]. The fetched data for the gas prices as well as token prices *i.e.,* cryptocurrency prices, record different measurements for these metrics. Among others, the Owloracle API data includes the `low` and `high` measures for a chosen timeframe. For the current evaluation, the timeframe is set to be one day. Hence, the `low` or `high` values indicate the lowest or highest values *e.g.,* gas or fiat price recorded on one particular day. For the upcoming evaluation of the historic gas and fiat price data, the `low` value is considered.

Based on the recorded historic gas as well as fiat prices, it is possible to calculate the transaction costs over time for the different solution smart contracts. Hence, any recorded

metric related to the gas used can be taken and multiplied by the historic gas and fiat prices for the respective chain. The process for the calculation remains the same, but the gas used value that is utilized is multiplied by the historic gas prices over time instead of the single gas price, which was recorded during the execution of the evaluation script. Based on these historic gas costs, the fiat costs can be calculated. For example, the average gas used per transaction for an evaluation run of a Deferral solution contract can be used to display the average gas and fiat prices over time for this contract.

After all, the historic price evaluation aims to accentuate how the values for gas prices and costs, as well as fiat prices and costs, must be treated with caution. The primary metric, which should remain the most consistent, is the gas used metric. The gas costs and fiat costs that are calculated based on these values should mainly serve as a point of reference for a better and more straightforward understanding of the results when it comes to the costs.

6.2.1.5 Evaluation Configuration and Results

All the generated result data, including the applied evaluation configurations, can be found in the `results` directory of the Deferral repository [132]. The `results` folder contains separate directories for the result data *i.e.,* `results/result-data` and the result visualizations and analysis *i.e.,* `results/result-visualizations`.

The results have been generated on the 5$^{\text{th}}$ of May 2023 by executing the evaluation scripts with 10, 100, and 500 participating users on the local Hardhat network. To calculate the fiat costs, all the fetched gas costs and gas usage values were converted into USD as a reference fiat currency. For the historic data, the timeframe was set to be one day. Thereby, gas and fiat price data were fetched for the time period from the end of January 2023 to the beginning of May 2023. For the analysis and evaluation, the lowest price entry on each day was used.

For all further details and configurations applied for the evaluation in this context, please refer to the evaluation scripts or the generated result data files in the Deferral repository [132].

6.2.2 Solution Contract Evaluation

The upcoming sections analyze the generated evaluation data and evaluate the individual results for all the Deferral solution contracts. Thereby, the different versions of the same design are evaluated against each other as well as compared to the previous design solutions and the corresponding contracts.

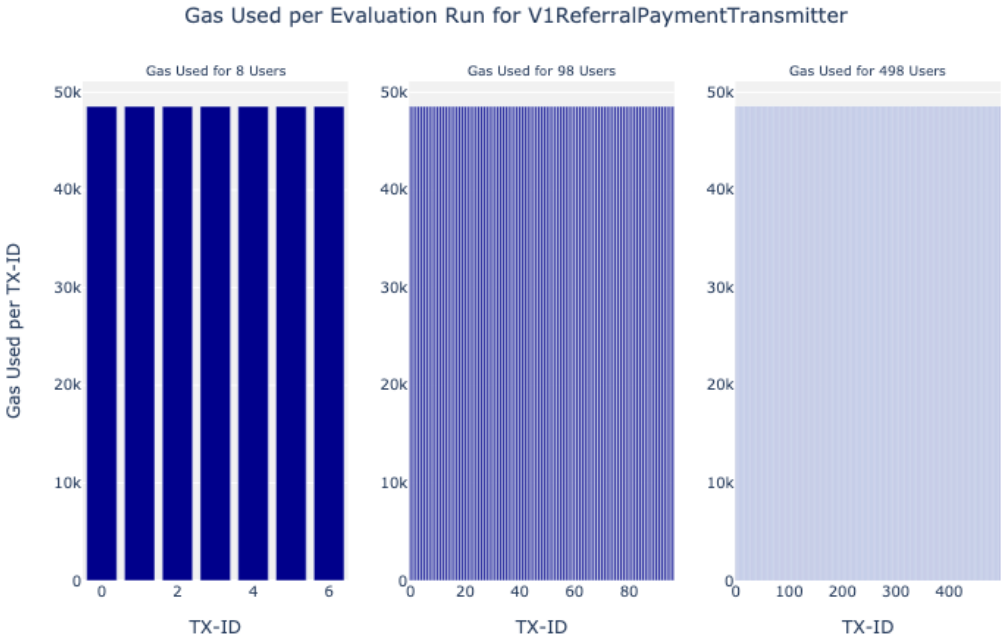


Figure 6.1: GasUsed Across the Evaluations of V1ReferralPaymentTransmitter

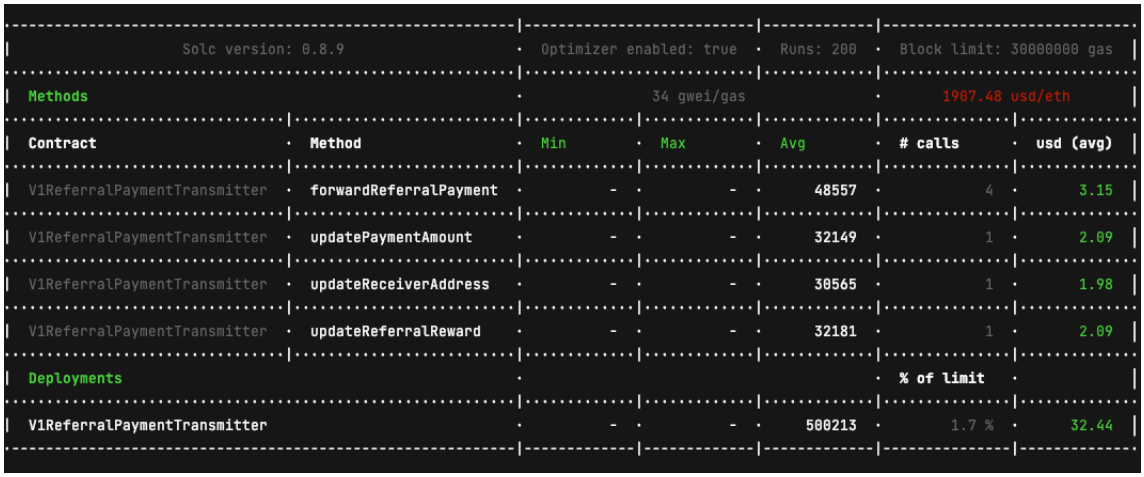
Nr. of Users	Avg gasUsed	Min gasUsed	Max gasUsed	Median gasUsed
8	48 555	48 545	48 557	48 557
98	48 556	48 545	48 557	48 557
498	48 555	48 533	48 557	48 557

Table 6.2: GasUsed for Evaluation Runs of V1ReferralPaymentTransmitter

6.2.2.1 Referral Payment Transmitter Contracts

Starting with version one of the referral payment transmitter contracts, Figure 6.1 displays the gas used per referral payment transaction within and across all three executed evaluation runs. Therein, it can be observed how the gas used remains consistent at around 50 000 gas among all referral transaction executions. Having a look at the calculated gas used metrics illustrated in Table 6.2 it shows how every executed referral payment transaction on the V1ReferralPaymentTransmitter contract fairly accurately uses the same amount of gas. Thus, the gas used is not affected by an increasing number of users and stays consistent.

With regard to the duration of the different referral transactions, the data in Figure D.1 does not show the same clear picture compared to the gas used. In general, the result data concerning the transaction duration often includes outliers where no clear explanations can be found. However, at a rougher estimate and with the exception of several outliers, the transaction duration underlines the trend for consistent transaction durations, which can be seen in the gas used results across the evaluation of version one of the referral payment transmitter contracts.



Solc version: 0.8.9	Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas
Methods	34 gwei/gas	1907.48 usd/eth	

Contract	Method	Min	Max	Avg	# calls	usd (avg)
V1ReferralPaymentTransmitter	forwardReferralPayment	-	-	48557	4	3.15
V1ReferralPaymentTransmitter	updatePaymentAmount	-	-	32149	1	2.09
V1ReferralPaymentTransmitter	updateReceiverAddress	-	-	30565	1	1.98
V1ReferralPaymentTransmitter	updateReferralReward	-	-	32181	1	2.09
Deployments					% of limit	
V1ReferralPaymentTransmitter		-	-	500213	1.7 %	32.44

Figure 6.2: Screenshot of the Gas Reporter Output for V1ReferralPaymentTransmitter

Chain	Gas Price in Gwei
BSC	3,0
Mainnet	78,975 115 466
Polygon-Mainnet	253,532 861 146
Arbitrum-Mainnet	0,1
Optimism-Mainnet	0,001
Avalanche	25,0
Goerli	1 076,663 332 944

Table 6.3: Evaluation Chain Gas Price Results for V1ReferralPaymentTransmitter

These results are backed by the Hardhat gas reporter output collected during the contract testing that is displayed in Figure 6.2. It shows the same average gas used for the referral payment transactions *i.e.,* for the `fowardReferralPayment` function. Furthermore, the gas reporter output displays the gas cost (34 Gwei per gas *cf.* Section 2.3.5) and the average cost of a referral transaction in USD (3,15) on the Ethereum blockchain. These Ethereum gas costs and fiat costs can also be observed in the generated result data. However, the generated result data displays a way higher average fiat cost on Ethereum per referral transaction of more than 7 USD, as it can be seen in Table D.4. This table outlines the metrics for the calculated transaction costs in USD recorded in the evaluation run with 500 *i.e.,* 498 participating users. This discrepancy is caused by the different gas prices and costs that were used for the calculations. Furthermore, as it can be assessed in Table 6.3, which illustrates the gas prices (in Gwei) recorded in the result data, the gas price for Ethereum *i.e.,* Mainnet (79 Gwei) is more than double the gas price shown in the gas reporter output (34 Gwei in Figure 6.2). These results demonstrate again why the results for the gas costs and fiat costs *e.g.,* costs in USD, must be treated and evaluated with caution.

For the upcoming evaluation sections focus lies on the collected and calculated gas used metrics. The collected gas and fiat prices and cost results are not displayed for every individual solution contract. Nonetheless, as the gas cost is the product of the gas prices and the gas used, these values can be inferred from the shown values and inspected in the result

Nr. of Users	Avg gasUsed	Min gasUsed	Max gasUsed	Median gasUsed
8	55 914	55 904	55 916	55 916
98	55 915	55 904	55 916	55 916
498	55 914	55 892	55 916	55 916

Table 6.4: GasUsed for Evaluation Runs of V3ReferralPaymentTransmitterUpgradable

data. The same goes for the fiat costs that are calculated based on the fiat prices and the gas costs. If required, these calculated result values for all individual smart contract evaluations can be inspected in the result data and visualization files [132]. As an overview of the prices in USD can be helpful, these values are attached in the Appendix in Section D.6 as well. Eventually, the gas and fiat price results are discussed later in this chapter during the overall evaluation for a few selected solution contracts (*cf.* Section 6.2.3). Consequently, based on the evaluation results for the V1ReferralPaymentTransmitter solution contract, it can be stated that the performance in terms of transaction cost and duration is not negatively affected by the increased volume of users participating in the system.

When it comes to the V3ReferralPaymentTransmitterUpgradable solution contract, the evaluation shows proportionate results to version one. The main difference is that for the amount of gas used per transaction, the results range slightly higher at around 56 000 gas, as Table 6.4 portrays. This increase in gas used is attributable to the upgrades pattern [97] that was implemented in version three of the referral payment transmitter contracts. Hence, the achieved flexibility in terms of contract deployment and implementation comes with a small but certain increase in transaction costs. Furthermore, the other results and visualizations for the V3ReferralPaymentTransmitterUpgradable contract show no further peculiarities. More results and visualizations of the evaluation data of version thee of the referral payment transmitter contract can be found in the Appendix in Section D.1 (*cf.* Figure D.2, D.3 and D.4, and Table D.5).

6.2.2.2 Referral Payment Quantity Evaluator Contracts

The results for the two referral payment quantity evaluator contracts show a different image compared to the referral payment transmitter design. First of all, Figure 6.3 depicts how the gas used for the execution is no more consistent with regard to the transactions within a single evaluation run. However, across all three evaluation runs, the gas used for the transactions evolves similarly. Thus, also for this solution design, the increased volume of participating users does show no negative effect on the gas used for the referral payment transactions.

Figure 6.4 shows the values are consistent if the gas used for transactions is analyzed per user iteration. The main finding is demonstrated in Figure 6.5 where the gas used is shown per user transactions. Therein, it can be seen how the first transactions of all the users are the most expensive in terms of gas used, followed by their third *i.e.,* last referral payment transaction. The second or the middle transaction, in this case, requires the least amount of gas.

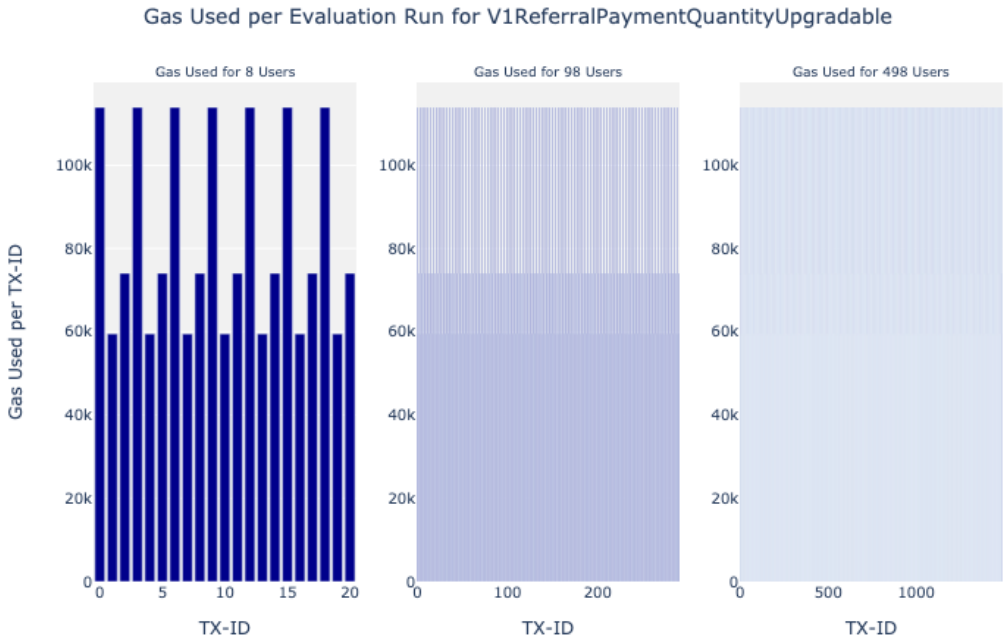


Figure 6.3: Gas Used per Transaction Across the Evaluations of V1ReferralPaymentQuantityUpgradable

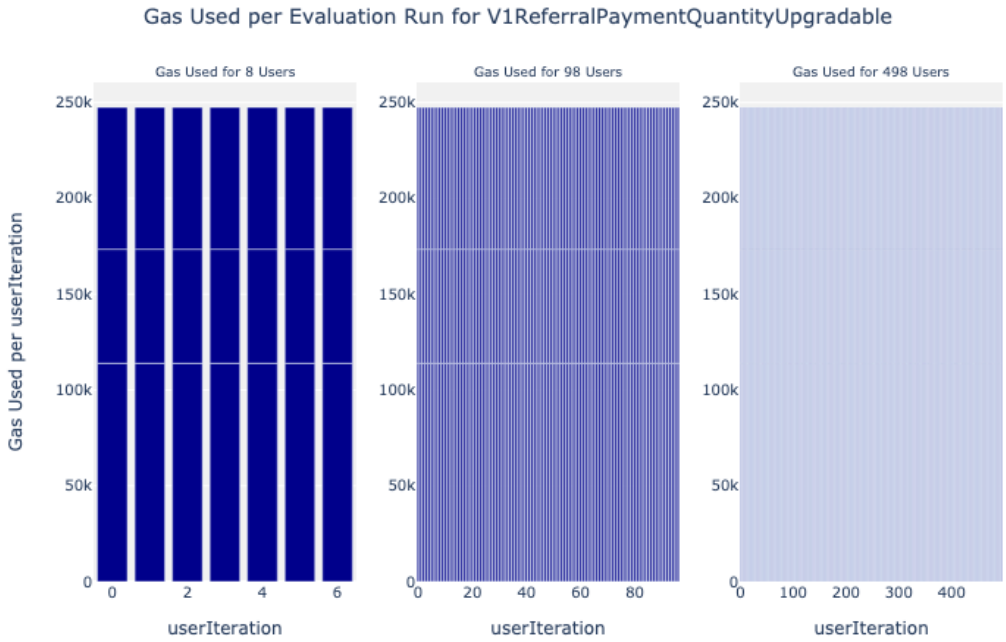


Figure 6.4: GasUsed per UserIteration Across the Evaluations of V1ReferralPaymentQuantityUpgradable

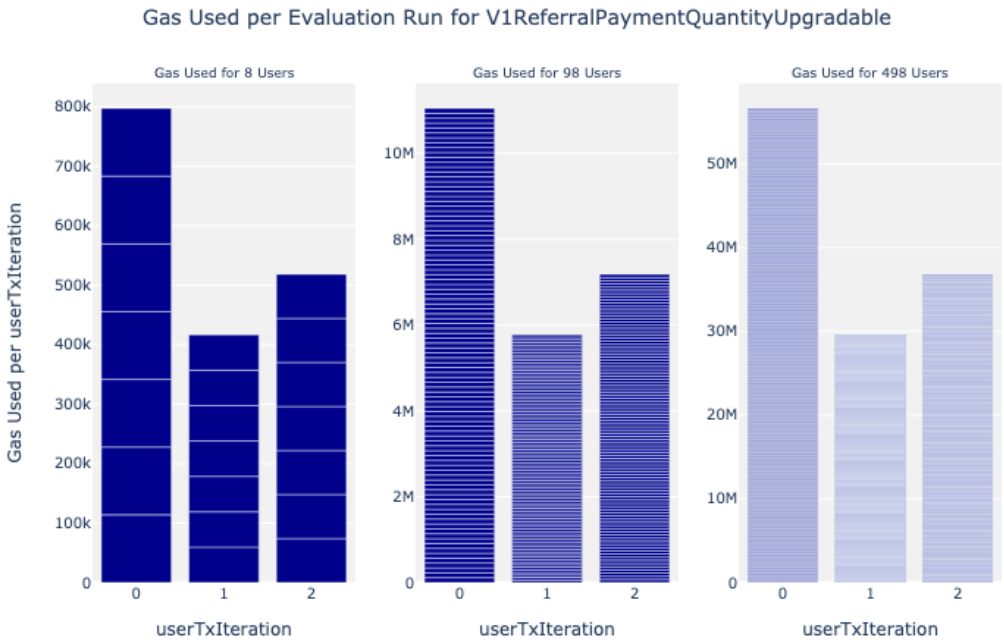


Figure 6.5: GasUsed per UserTxIteration Across the Evaluations of V1ReferralPaymentQuantityUpgradable

Nr. of Users	Avg gasUsed	Min gasUsed	Max gasUsed	Median gasUsed
8	82 484	59 508	113 900	74 038
98	82 485	59 508	113 900	74 038
498	82 484	59 496	113 900	74 038

Table 6.5: GasUsed for Evaluation Runs of V1ReferralPaymentQuantityUpgradable

A closer look at the implementation and flow of the referral processes for the referral payment quantity evaluator contracts clarifies these differences. On the one hand, with the first referral payment transaction, the user and corresponding referral process data are registered *i.e.,* stored on the smart contract. Smart contract transactions or EVM operations that change any value from a zero to any other non-zero value are more expensive than operations where a value's zeroness is not altered [148]. The gas used values for smart contract operations on the EVM and a fee schedule overview can be found in the Ethereum yellow paper [148]. Thus, the initial storage of the user referral process data in the first referral payment transaction of every user is more expensive in terms of gas used than the second transaction, where values are only updated *i.e.,* their zeroness is not changed.

On the other hand, the disparity in gas used between the second and the third user transaction (*cf.* Figure 6.5) is caused by the completion of the referral process within the third transaction and the resulting distribution of the referral rewards. The sending of the rewards causes extra costs in terms of gas used compared to only updating the referral process data in the second transaction. It is interesting to see that the first transaction,

Nr. of Users	Avg gasUsed	Min gasUsed	Max gasUsed	Median gasUsed
8	81 865	58 850	113 242	73 499
98	81 866	58 850	113 242	73 499
498	81 866	58 838	113 242	73 499

Table 6.6: GasUsed for Evaluation Runs of V2ReferralPaymentQuantityUpgradable

which initiates the storage of the referral process data for the users, is the most expensive in this case.

The small differences in the implementation approaches used for versions one and two of the referral payment evaluator contracts that were introduced in Section 5.2.2 can be spotted in Tables 6.5 and 6.6. Comparing the two tables shows that the average gas used for version two of the contracts is marginally smaller. The outputs of the gas reporter for the two contracts confirm this evaluation result (*cf.* Figures D.5 and D.6 in the Appendix in Section D.2).

Furthermore, the remaining values shown in the gas reporter outputs, especially the average gas used, are not really comparable in this case since the configuration of the tests that is essential for the results for the gas reporter differs from the configuration and parameters applied in the evaluation scripts. After all, neglecting the readability of the implemented Solidity code and writing bigger but fewer functions on the contract brings a small advantage in terms of gas used. Whether this small cost advantage is worth the compromises of the code readability or not is debatable.

With regard to the duration metrics, there are no specificities or trends that can be identified. Eventually, the functionality that is added to the payment quantity evaluator referral systems compared to the referral payment transmitter solutions can be well observed in the presented results in terms of gas used. Further results and visualizations of the evaluation data of the referral payment quantity evaluator contracts can again be found in the Appendix in Section D.2 or in the `results` folder in the Deferral repository [132].

6.2.2.3 Referral Payment Value Evaluator Contracts

The results for the V1ReferralPaymentValueUpgradable referral contract are very similar to the results that were discussed for the V2ReferralPaymentQuantityUpgradable contract. Figure 6.6 illustrates again the differences in terms of gas used between the initial referral payment transactions and the final transactions. This time, as the referral process requires four transactions per user to be completed, it can be observed how all other transactions that do not initiate or complete a user's referral process utilize a consistent amount of gas.

Considering the V2ReferralPaymentValueUpgradable contract, the evaluation results show that the approach of claimable rewards and the added functionality cause additional costs in terms of gas used for the final transaction, which completes the referral process (*cf.* Tables D.1 and D.2 in the Appendix Section D.3)). As has been shown before, initializing

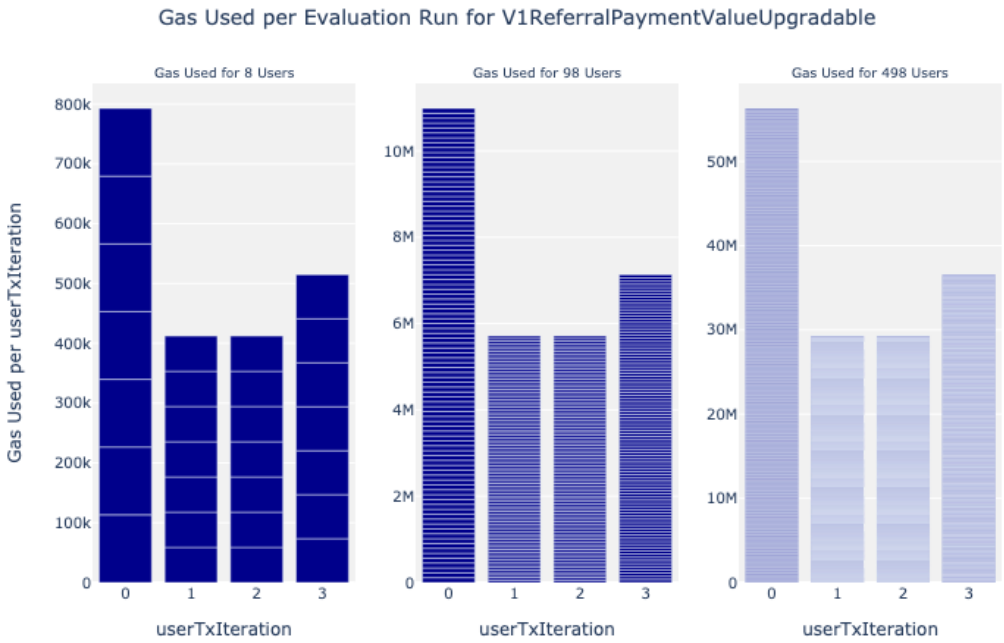


Figure 6.6: GasUsed per UserTxIteration Across the Evaluations of V1ReferralPaymentValueUpgradable

or storing new data, that in this case is done by keeping track of the claimable rewards for users, utilizes more gas than directly sending and distributing the rewards within the final transaction [148]. Hence, the design and implementation of version two of the referral payment quantity evaluator contracts require more gas than version one. To be more precise, the transaction that completes the referral process requires more gas in version two of the contract than in version one (*cf.* Figure 6.7). This is also noteworthy since, in the evaluation process of the V2ReferralPaymentValueUpgradable contract, the users have completed the referral process but have not yet received the rewards, as they must be claimed within an additional transaction.

The results for the V3ReferralPaymentValueUpgradable contract show essentially the same picture as the V2ReferralPaymentValueUpgradable results when they are compared to the results for version one. Again, as the final referral payment transaction completes the referral process and, this time, must distribute two-sided referral rewards *i.e.,* two send operations to two different users, it requires more gas (*cf.* Figure 6.8).

Comparing version three to version two, it shows that the claimable rewards functionality in the final transactions of version two utilizes more than 600 000 gas (*cf.* Figure 6.7) in contrast to the two-sided reward distribution in the complete transaction of version three which uses less than 600 000 gas (*cf.* Figure 6.8). Consequently, the design including claimable rewards in version two utilizes the most gas, although the users have not received the rewards yet. The transaction for claiming rewards would add additional costs to the user side (*cf.* Figure D.13) that have not been considered in the evaluation process.

These findings also align with the results shown in the gas reporter outputs (*cf.* Figures D.10, D.13 and D.16 in the Appendix in Section D.3). Nonetheless, the exact values

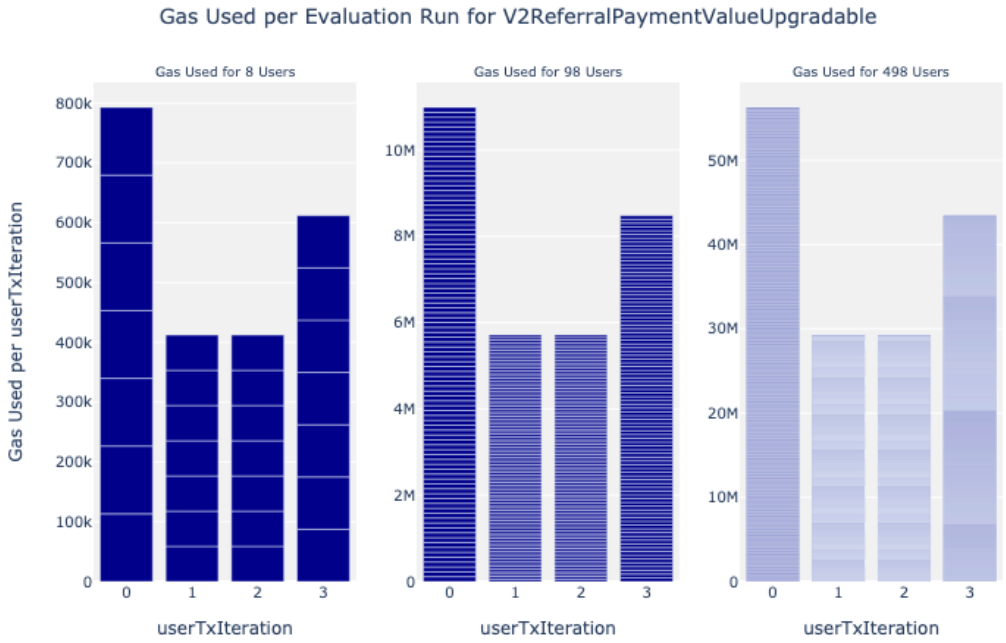


Figure 6.7: GasUsed per UserTxIteration Across the Evaluations of V2ReferralPaymentValueUpgradable

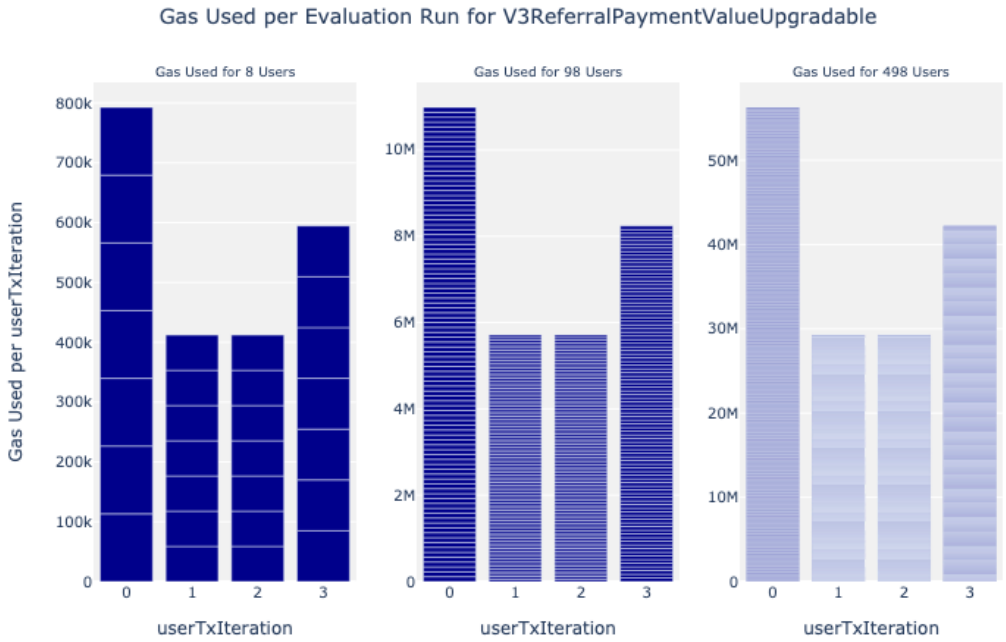


Figure 6.8: GasUsed per UserTxIteration Across the Evaluations of V3ReferralPaymentValueUpgradable

Nr. of Users	Avg gasUsed	Min gasUsed	Max gasUsed	Median gasUsed
7	95 711	61 800	136 686	108 990
97	303 457	61 800	1 383 025	116 302
497	1 226 711	61 800	6 922 694	116 302

Table 6.7: GasUsed for Evaluation Runs of V1ReferralMultilevelRewardsUpgradable

in the gas reporter outputs can again not be compared as the configuration and parameters applied in the evaluation scripts differ from the values used in the tests.

In conclusion, for the referral payment value evaluator solutions, it can be stated that they are not affected by the increasing volume of participating users. The tracked and evaluated transaction durations also did not bring any special insights. Moreover, the added functionalities related to the reward distribution and allocation come with additional costs (*cf.* Tables in the Appendix in Section D.3). Therein, the two-sided reward allocation functionality is more cost-efficient than the claimable rewards functionality as it does not require storing additional values on the smart contract.

6.2.2.4 Referral Payment Multilevel Reward Evaluator Contracts

The evaluation results for the two referral payment multilevel reward evaluator contracts reveal various insights. As this solution design is more complex than the previously discussed Deferral solutions, it could be expected that the average costs in terms of gas used would be higher for these contracts.

This is confirmed in Table 6.7 that shows an average gas used value of around 96 000 gas for the evaluation run with 10 *i.e.,* 7 involved users. Over and above, Table 6.7 illustrates the big differences in gas used metrics between the three evaluation run. Therein, the average and maximum gas used metrics are increasingly higher the more users are involved in the referral system.

By having a look at the gas used for the individual transactions within the three evaluation runs in Figure 6.9, it can be observed how every third transaction is increasing in cost the more users are involved. Thus, especially in the evaluation runs with 100 and 500 users, the distributions of the gas used per transaction are skewed extremely negatively. The same pattern shows for the duration of the transactions, which are depicted in Figure 6.10.

As a result, the gas used and the duration of the referral payment transactions are higher for users that participate later in the referral system. This trend is also demonstrated in Figures 6.11 and 6.12 which show the gas used and duration per transactions per user.

For instance, it takes longer and is more expensive for user (10) than for user (1) to complete the referral process on the V1ReferralMultilevelRewardsUpgradable contract. In the case of a couple of hundreds of users who are participating and completing the referral process, the difference in terms of gas used for the final transaction of a user to complete the process can grow to several millions and would grow even further with more participating users (*cf.* Figures 6.11 and 6.12).

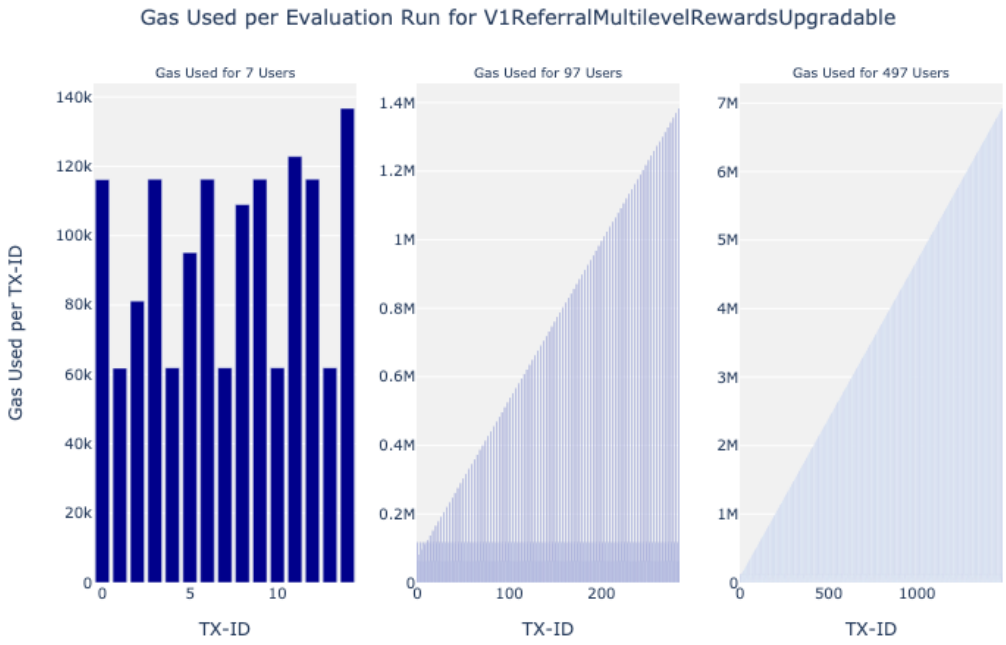


Figure 6.9: GasUsed per Transaction Across the Evaluations of V1ReferralMultilevelRewardsUpgradable

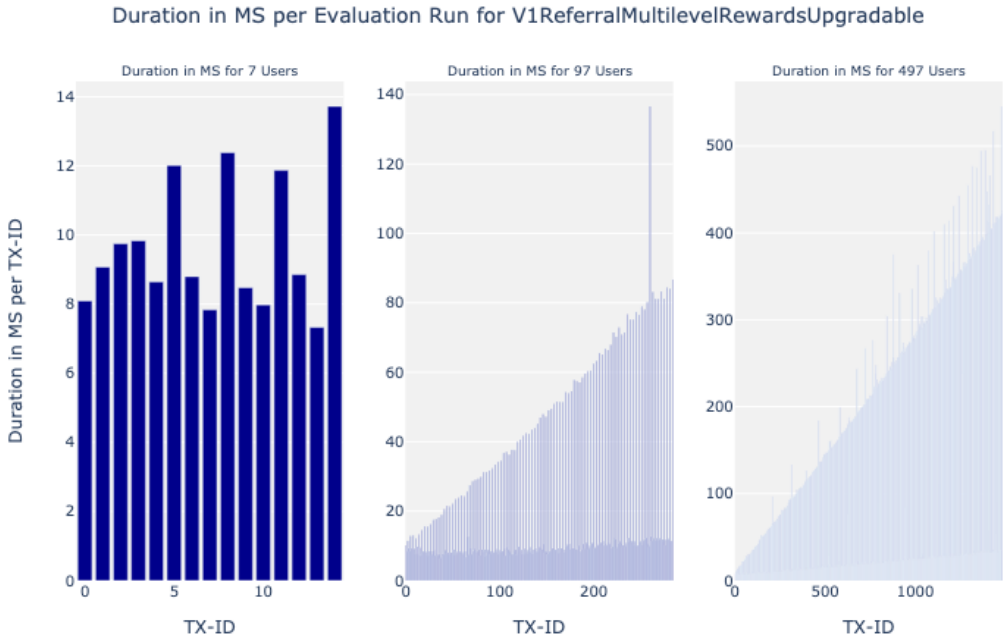


Figure 6.10: DurationInMs per Transaction Across the Evaluations of V1ReferralMultilevelRewardsUpgradable

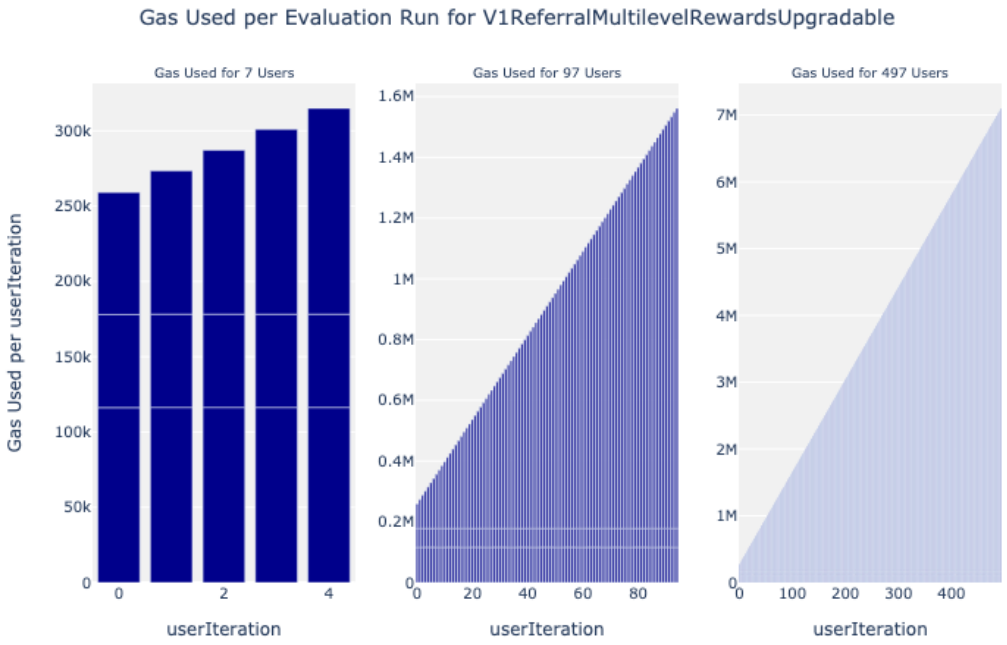


Figure 6.11: GasUsed per UserIteration for the Evaluations of V1ReferralMultilevelRewardsUpgradable

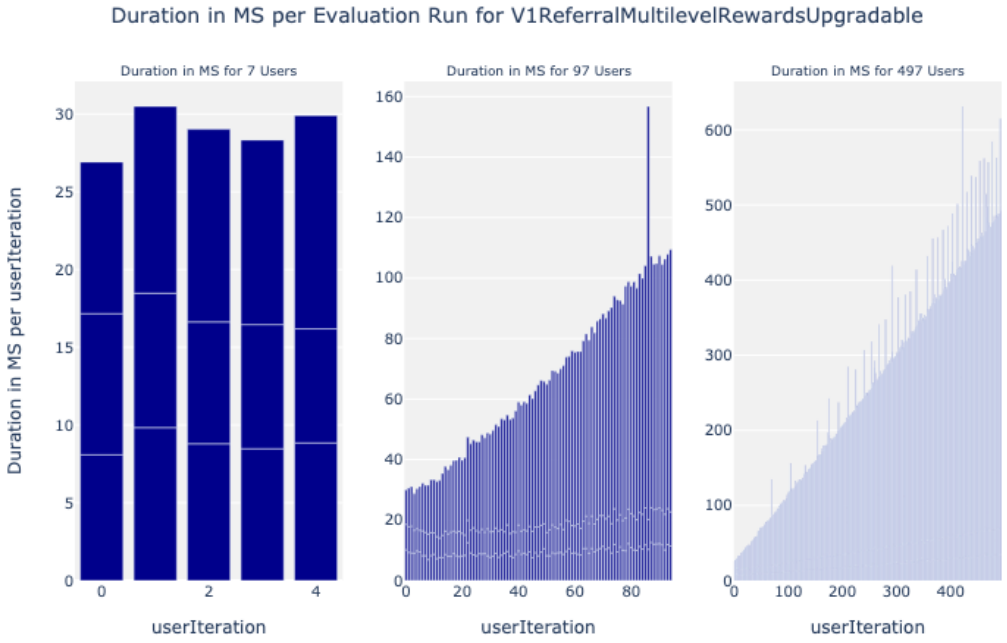


Figure 6.12: DurationInMs per UserIteration for the Evaluations of V1ReferralMultilevelRewardsUpgradable

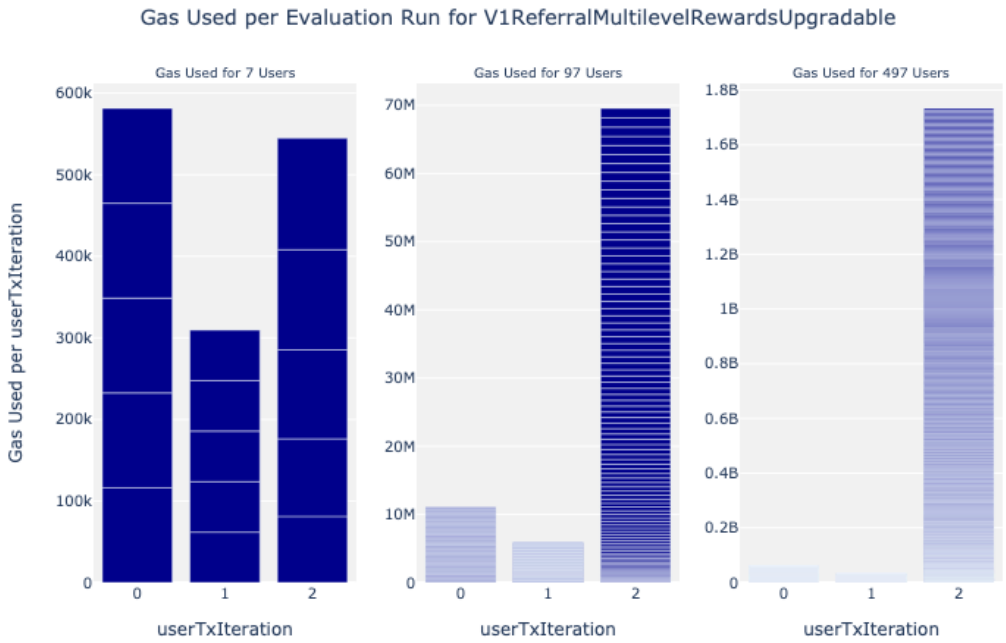


Figure 6.13: GasUsed per UserTxIteration Across the Evaluations of V1ReferralMultilevelRewardsUpgradable

The results of the evaluation of the V1ReferralMultilevelRewardsUpgradable referral system clearly show the impact of the multilevel reward distribution functionality that was added. As outlined before in Sections 4.4.2.4 and 5.2.4 the contract splits the rewards and distributes the assets to all related referrers in the referral chain. In the evaluation process for this contract, all involved users refer each other and thus create a referral chain with the length of the number of users involved in the evaluation process. Hence, within the final transaction of user (1), the rewards must only be sent to one previous referrer. However, for user (10), there are already ten previous referrers that receive rewards. As a result, the final transaction of user (10) requires more gas since the assets must be sent to ten different other users. Due to this functionality, the costs and durations of the completed transactions for the participants increase with higher volumes of users. Further, these transactions take longer to execute as more send operations have to be performed by the contract. This trend is well outlined in Figures 6.11 and 6.12 which show the gas used and durations grouped per user.

Apart from this, it is interesting to analyze the gas used and duration per transaction grouped by the user transaction iteration, as it is outlined in Figures 6.13 and 6.14. In Figure 6.13, the same pattern that has been shown in the referral payment quantity and value evaluators can be identified. For the evaluation run with 10 *i.e.,* 7 users, the initial transactions that start the referral process require more gas than the final transactions that complete the referral process and distribute the rewards. However, in the same graph with 7 users, it can also be recognized how the stacked bars for the third transaction increasingly get bigger with higher numbers of users. Thereafter, in the chart for the evaluation run with 100 *i.e.,* 97 users, the gas used per transaction iteration of the third

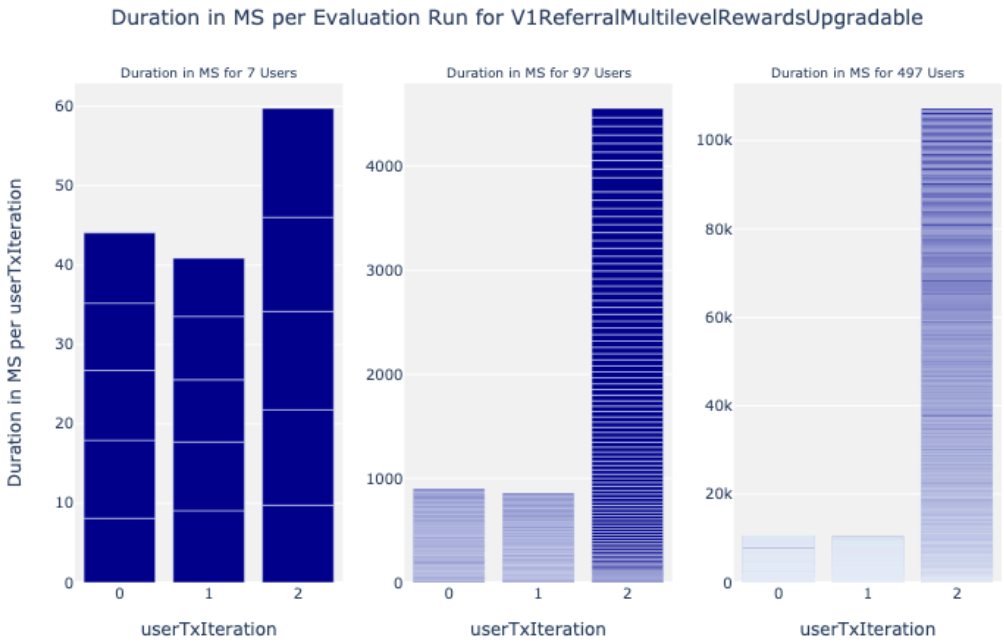


Figure 6.14: DurationInMs per UserTxIteration Across the Evaluations of V1ReferralMultilevelRewardsUpgradable

and final transactions has already noticeably outgrown the gas required for the initial transaction. This result is further underlined in the visualization for the evaluation run with 500 *i.e.,* 497 users, where the bars for the first and second transactions are barely visible anymore compared to the costs in terms of gas used for the final transactions. This illustrates how the cost of distributing reward assets can surpass the gas costs required for storing or initiating the referral process data when there are multiple users to whom referral rewards must be distributed.

Eventually, it can be declared that the V1ReferralMultilevelRewardsUpgradable contract is significantly affected by the volume the referral system has to handle. More users participating in the referral system imply more costs and thus has a negative effect on the cost as well as the performance in terms of the duration.

For version two of the referral payment multilevel rewards contract, two additional features have been included compared to version one (*cf.* Sections 4.4.2.4 and 5.2.4). The effect of the maximum referral reward level that could be defined to set a limit on the length of the referral chain for the reward distribution can clearly be identified in the results of version two if they are compared to the results of version one.

First, Table 6.8 outlines how the metrics for the gas used are again more consistent across the different evaluation runs compared to the metrics for version one in Table 6.7. The same finding appears in Figure 6.15, which shows the gas used for the different transactions within the different evaluation runs.

Furthermore, on closer inspection of Figure 6.16, it clarifies that the transaction costs

Nr. of Users	Avg gasUsed	Min gasUsed	Max gasUsed	Median gasUsed
7	100 338	64 594	125 914	118 974
97	103 088	64 594	125 914	119 096
497	103 211	64 594	125 914	119 096

Table 6.8: GasUsed for Evaluation Runs of V2ReferralMultilevelRewardsUpgradable

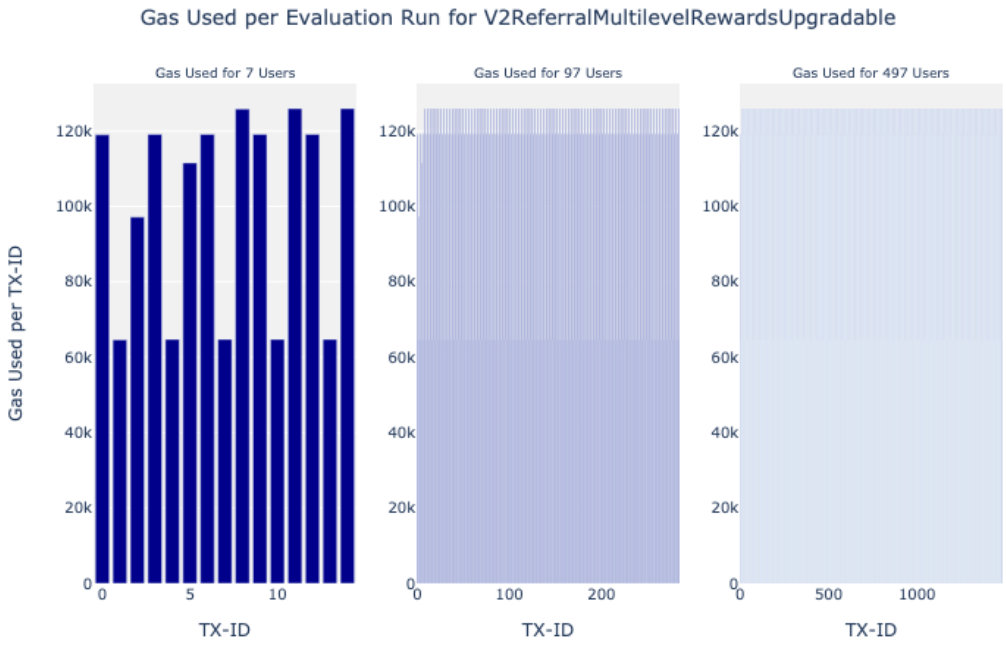


Figure 6.15: GasUsed per Transaction Across the Evaluations of V2ReferralMultilevelRewardsUpgradable

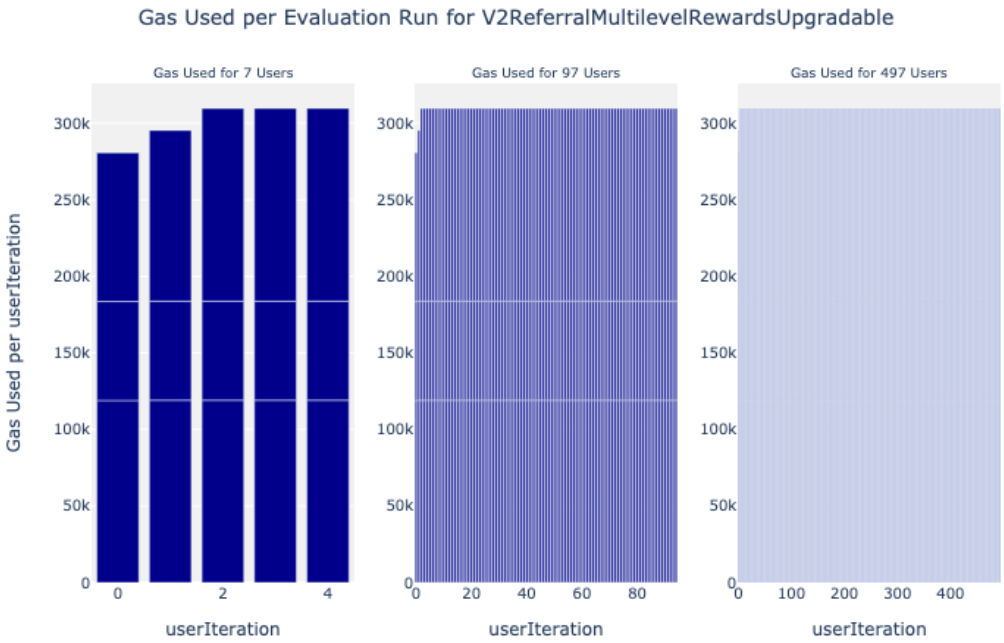


Figure 6.16: GasUsed per UserIteration for the Evaluations of V2ReferralMultilevelRewardsUpgradable

per user are consistent and the same again for almost all participating users. Since the maximum reward level is set to three for the evaluation of this contract, the rewards are distributed to three prior referrers at most. The reward chain is still built the same way and includes all participating users. However, the reward distribution is limited in this version of the contract. Thus, within every final transaction of all users, the rewards must be sent to three prior referrers. The only exceptions are the first two users. For these two users, their referral chain includes only one or two previous referrers. As a result, their final transactions require slightly less gas than all the other users with a referral chain of three or more users. This pattern can be spotted in Figure 6.16 as well as in Figure 6.17.

Moreover, in the evaluation run with 10 *i.e.,* 7 users in Figure 6.17, it can be observed how the gas used for the final transaction that distributes the rewards exceeds the costs of the initial transaction that stores the referral process data. This is also the case in the other two evaluation runs. Nonetheless, it can not be recognized in these visualizations by the bare eye. On closer examination of the stacked bars, it can be seen that the final transaction of user (1), which has to distribute rewards to only one referrer, is cheaper in terms of gas used than the initial storage transaction of user (1). For the second user, this is still the case. Nevertheless, the final transaction uses already basically the same amount of gas as the initial transaction in the case of user (2). The exact costs in terms of gas used for the first four users are also displayed in Table 6.9. For user (3), it can be observed that the final transaction requires more gas than the initial transactions. The same applies to user (4) and to all further users since, for all these users, the referral rewards are sent to three previous referrers. Consequently, this case outlines well how the distribution of the rewards can require more gas than the initiation of the referral process data if the rewards must be distributed to multiple other users. Another detail to mention at this point is

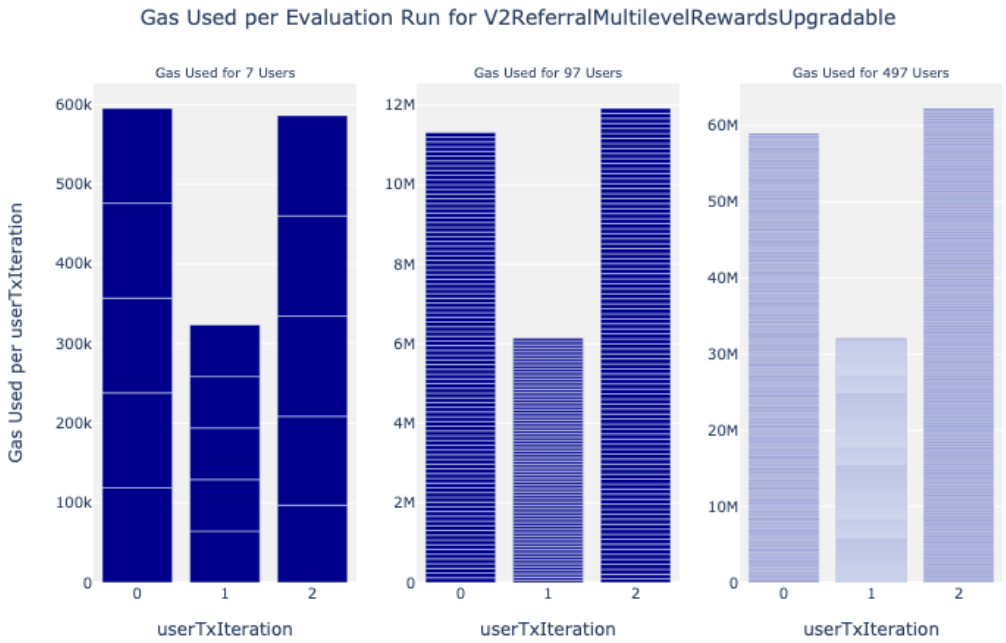


Figure 6.17: GasUsed per UserTxIteration Across the Evaluations of V2ReferralMultilevelRewardsUpgradable

User ID	Initial TX gasUsed	Second TX gasUsed	Final TX gasUsed
User 1	118 974	64 594	97 133
User 2	119 096	64 716	111 529
User 3	119 096	64 716	125 803
User 4	119 084	64 704	125 902

Table 6.9: GasUsed per Transaction for the first 4 Users in the Evaluation run with 10 *i.e.,* 7 Users for V2ReferralMultilevelRewardsUpgradable

that the V2ReferralMultilevelRewardsUpgradable also reintroduces the functionality for two-sided rewards. Hence, within the final transaction, the rewards are not only sent to a maximum of three previous referrers but also to the referee. Therefore, the final transactions from user (3) onward perform a total of four send operations for the reward distribution. Thus, as can be observed for user (2) and user (3) in Table 6.9, the threshold where the final transaction becomes more expensive than the initial transaction in this context lies between three send operations for the final transaction of user (2) and four send operations for the final transaction of user (3).

For the two multilevel reward evaluator contracts, the discussed findings from the evaluation scripts are not confirmed by the gas reporter outputs (*cf.* Figures D.19 and D.20 in the Appendix in Section D.4). This can mainly be attributed to two reasons. First, as mentioned before, the gas reporter differs from the configuration and parameters utilized in the evaluation scripts. Hence, in the tests, the length of the created referral chain is not comparable to the one in the evaluation scripts, especially for higher volumes of users. Second, the gas reporter evaluates the gas used metrics for every called function

individually. Therefore as the multilevel reward contracts implement function overloading (*cf.* Section 5.2.4.1), the results for referral payment transaction executions calling the `registerReferralPayment` function are separated (*cf.* Figures D.19, and D.20)

Ultimately the evaluation results of the V2ReferralMultilevelRewardsUpgradable referral system outline how the introduction of a maximum reward level for the distribution of rewards can significantly reduce the negative effects caused by high volumes of participating users compared to version one of the referral payment multilevel rewards solution design.

6.2.2.5 Referral Payment Multilevel Token Reward Evaluator Contracts

The main distinction between the V1ReferralMultilevelTokenRewardsUpgradable contract compared to the V2ReferralMultilevelRewardsUpgradable discussed before has been introduced in Sections 4.4.2.5 and 5.2.5.1. The difference of using an ERC20 as reward currency in this referral system instead of native crypto assets also shows in the results.

As illustrated in Table 6.10, the cost metrics in terms of gas used are higher compared to the values in Table 6.8. These results could be expected as it is generally more expensive to transfer and send assets of any ERC20 token compared to native cryptocurrency assets of any EVM-based blockchain. Transferring an ERC20 token involves interacting with a smart contract, which adds computational effort compared to a simpler transfer of native assets. When distributing or sending an ERC20 token, a smart contract must execute several functions to update the token balances of the sender and the receiver and perform additional checks to assess the validity of a transfer.

The results of the evaluation of the V1ReferralMultilevelTokenRewardsUpgradable display basically the same patterns as the results for version two of the referral payment multilevel reward evaluator discussed before. The same applies to the gas reporter outputs. These results and visualizations can be found in the Appendix in Section D.5.

6.2.3 Overall Evaluation

After all the evaluation results for the individual Deferral smart contracts and the different solution designs have been talked over, the overall view of the evaluation results can be considered.

So far, it showed how with increasing complexity and functionality, the costs concerning

Nr. of Users	Avg gasUsed	Min gasUsed	Max gasUsed	Median gasUsed
7	125 878	89 471	161 073	143 961
97	126 617	89 471	161 073	143 973
497	126 650	89 471	161 073	143 973

Table 6.10: GasUsed for Evaluation Runs of V1ReferralMultilevelTokenRewardsUpgradable

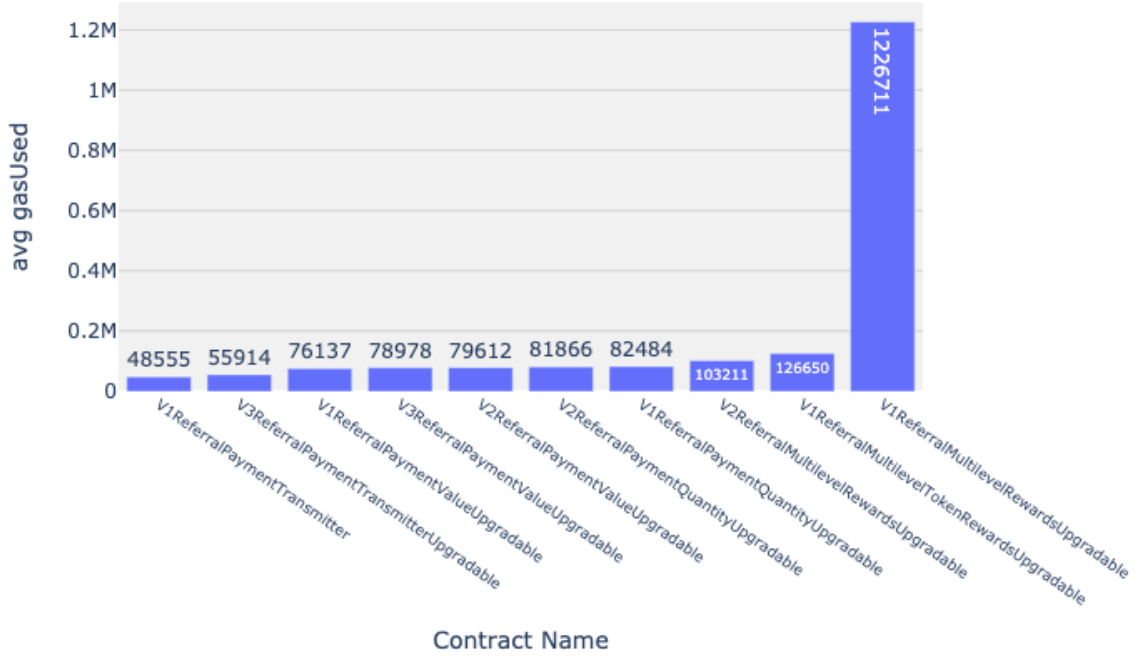


Figure 6.18: Overall Average Gas Used per Transaction Across All Deferral Contracts

gas used for referral payment transactions on the Solidity smart contracts increased. In this context, Figure 6.18 gives an overview of this trend by displaying the average gas used per transaction for all Deferral solution contracts regarding the evaluation runs done with 500 users. In this figure, the discussed findings bear out again. It can be observed how the average gas used per transaction is the lowest value for the referral payment transmitter design, as these contracts provide the simplest versions for referral systems. Thereon, the results are very similar and close for all versions of the referral payment quantity and referral payment value evaluator contracts. Eventually, the multilevel rewards and multilevel token rewards transactions require the most gas. This is no surprise, given that they integrate functionalities from both the referral payment value and quantity evaluator contracts while also facilitating multilevel reward distributions. At this point, it is important to keep in mind that these results, as well as all other presented results, can be subject to change in further evaluations if the contract configurations are adjusted *e.g.,* the referral conditions defined on the contracts are changed in the evaluation scripts.

In addition, Table 6.11 illustrates the gas used for the initial deployment of the different contracts. Therein, a similar picture shows that larger and more complex contracts which implement more functionality require more gas to be deployed. Furthermore, Figure 6.18 emphasizes again that version one of the multilevel reward contracts is the only solution that shows negative effects when dealing with a high volume of users. The figure shows the most extreme case as it depicts the evaluation run with the highest volume of users. In this case, the V1ReferralMultilevelRewardsUpgradable contract, on average, roughly requires ten times the gas that is used by the second most solution contract. However, only concerning deployment costs (*cf.* Table 6.11), this cannot be seen since, for the deployment of a contract, the impact of high volumes of users does not show.

Deferral Contract Deployments	GasUsed
V1ReferralPaymentTransmitter	500 213
V3ReferralPaymentTransmitterUpgradable	554 099
V1ReferralPaymentValueUpgradable	690 041
V2ReferralPaymentQuantityUpgradable	690 041
V1ReferralPaymentQuantityUpgradable	750 116
V2ReferralPaymentValueUpgradable	741 038
V3ReferralPaymentValueUpgradable	777 322
V1ReferralMultilevelRewardsUpgradable	1 030 918
V2ReferralMultilevelRewardsUpgradable	1 249 422
V1ReferralMultilevelTokenRewardsUpgradable	1 411 504

Table 6.11: Gas Reporter Output Showing GasUsed for the Deferral Contract Deployments

6.2.3.1 Gas and Fiat Costs Evaluation

Furthermore, the evaluation results across all the solution smart contracts can be reviewed, focusing on the gas costs required on the different evaluation chains and the resulting fiat costs in USD for the calculated gas used metrics. These results can also be found for all the individual contracts either in the Appendix in Section D.6 or in the result data [132].

In that regard, Figure 6.19 illustrates the gas cost calculated based on the recorded gas used values (*cf.* Figure 6.18). It includes data for all solution contracts across all evaluation chains coming from the evaluation run results where the referral processes were executed with 500 users. Moreover, in Figure 6.19, the V1ReferralMultilevelRewardsUpgradable contract, as well as the gas costs for the Goerli testnet, are omitted for better visibility. The two visualizations, including these values, can be found in the Appendix in Section D.7(*cf.* Figures D.30 and D.31). Eventually, Table D.14 displays the same gas cost values as numbers.

By observing the results shown in Figure 6.19, it clarifies which blockchains have the highest gas costs among the chosen evaluation chains. Therein, Polygon has the highest gas costs in Gwei, followed by the Ethereum Mainnet. On the other end, the Optimism chain implies the lowest gas costs before the Binance Smart Chain (*cf.* Table D.14). As mentioned before (*cf.* Section 6.2.1.1), the gas cost results can be misleading. They are based on gas prices and cannot be compared with each other as they represent the respective value in the native cryptocurrency of the corresponding blockchain. Consequently, to reduce these values to a common denominator again, the calculated fiat costs in USD must be examined.

The actual costs in USD show a different picture compared to the gas costs but differ significantly across the different evaluation blockchains as well. Within Figure 6.20, the V1ReferralMultilevelRewardsUpgradable contract is again excluded for better visibility. The unfiltered visualization (*cf.* Figure D.32) is attached in the Appendix together with Table D.15 which portrays the exact fiat cost values in USD as numbers. The results in Figure 6.20 demonstrate how the Ethereum Mainnet is by far the most expensive out of all the selected evaluation chains. The difference is too big to observe the results of the

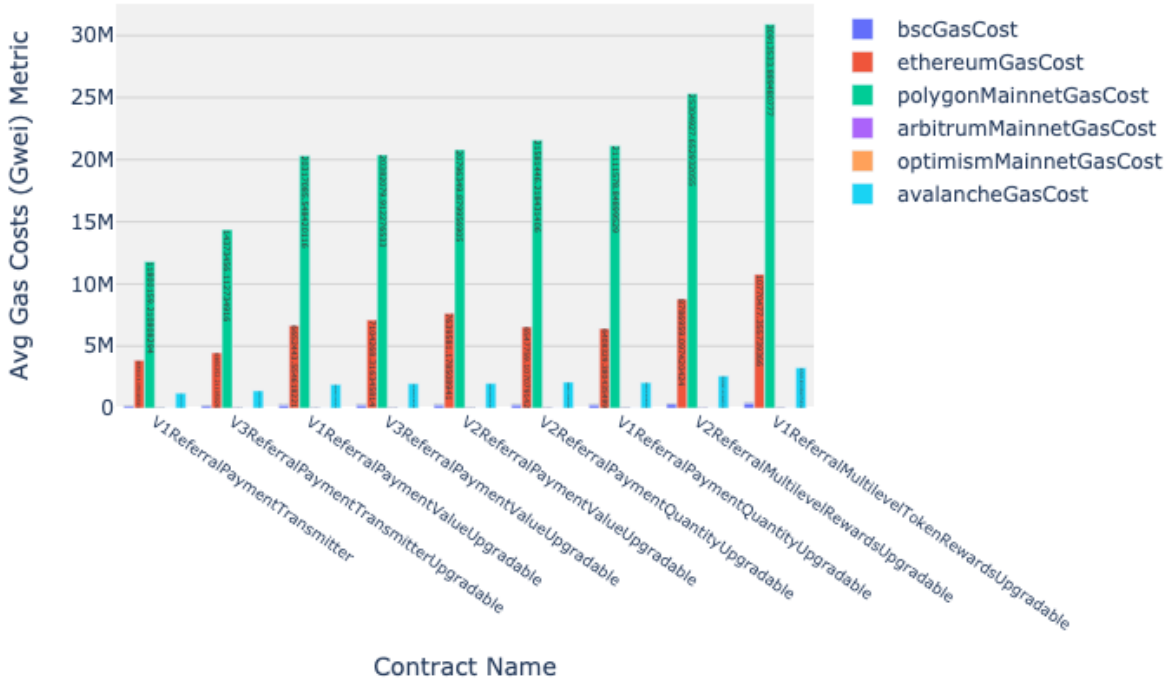


Figure 6.19: Filtered Overall Average Gas Costs in Gwei per Contract per Evaluation Chain

other evaluation chains in Figure 6.20. Consequently, the fiat prices in USD have been outlined again in Figure 6.21 without the Ethereum chain.

By analyzing Figure 6.21 or the rounded values in Table D.15, the Binance Smart Chain can be identified as the second most expensive chain, followed by Avalanche, which is only about half as expensive as Binance. Thereafter, the results of all three layer-2 evaluation chains confirm their reputation as cheaper alternatives. Polygon, which has had the highest gas costs in Gwei before (*cf.* Figure 6.19), is the most expensive chain before Arbitrum and eventually Optimism. Further, even if the Goerli testnet is not a relevant case for investigating the real-world feasibility of the solutions since Goerli Ethers have no real monetary value, it is still interesting to see that the testnet would be more expensive than the Optimism layer-2 solution. Moreover, the rounded fiat costs on Optimism (*cf.* Table D.15) show zero for all Deferral solution contracts but for the most expensive V1ReferralMultilevelRewardsUpgradable contract. For this contract, the average rounded transaction costs are displayed as 0,002 USD. For comparison, the average costs for the same contract on Ethereum are over 200 USD.

Eventually, Figure D.32 and Table D.15 highlight two main aspects. First, the costs in USD of version one of the multilevel reward contract increase significantly, with more users participating compared to the other solution contracts. And second, Ethereum is by far the most expensive blockchain solution concerning transaction costs in USD for all the Deferral referral systems.

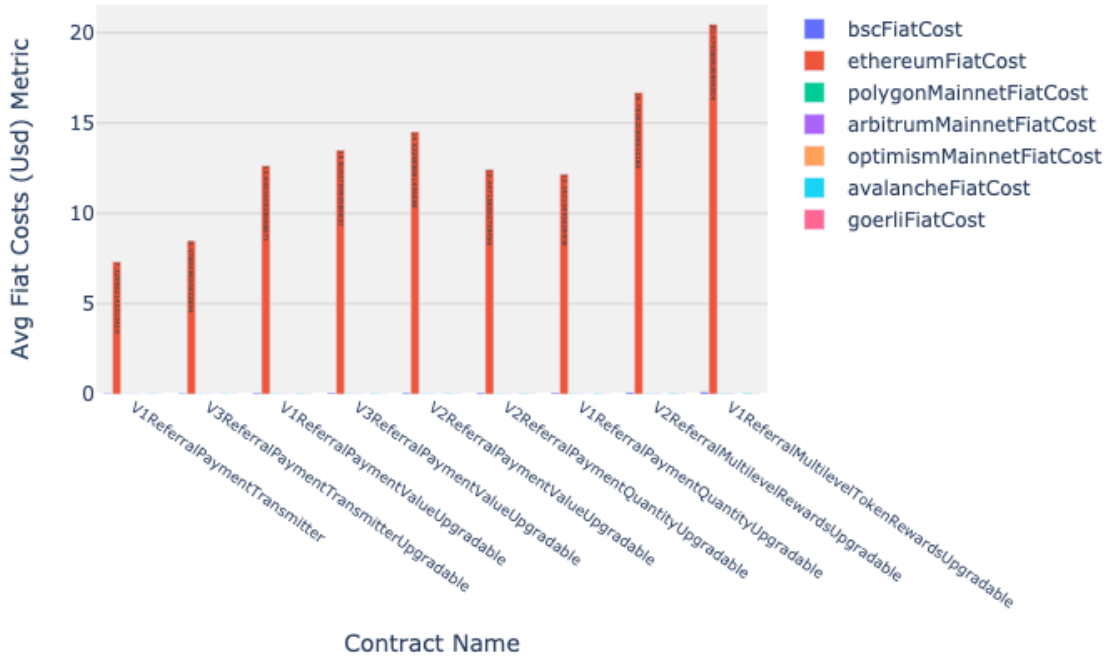


Figure 6.20: Filtered Overall Average Fiat Costs in USD per Contract per Evaluation Chain

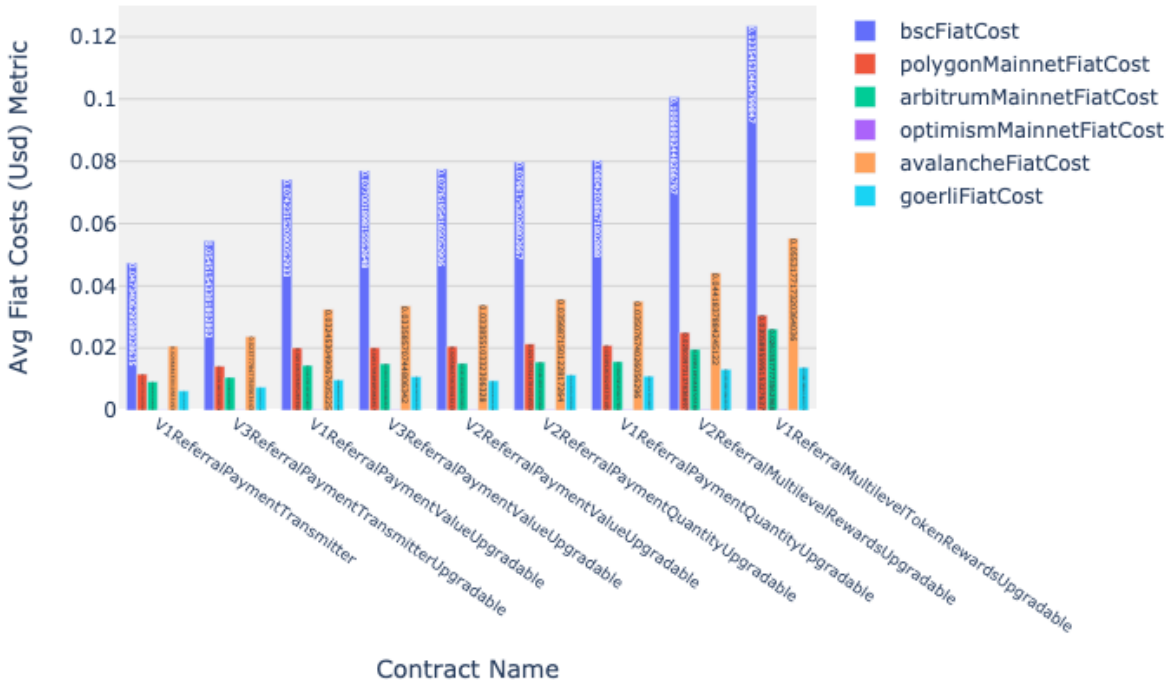


Figure 6.21: Filtered Overall Average Fiat Costs in USD per Contract per Evaluation Chain Without Ethereum

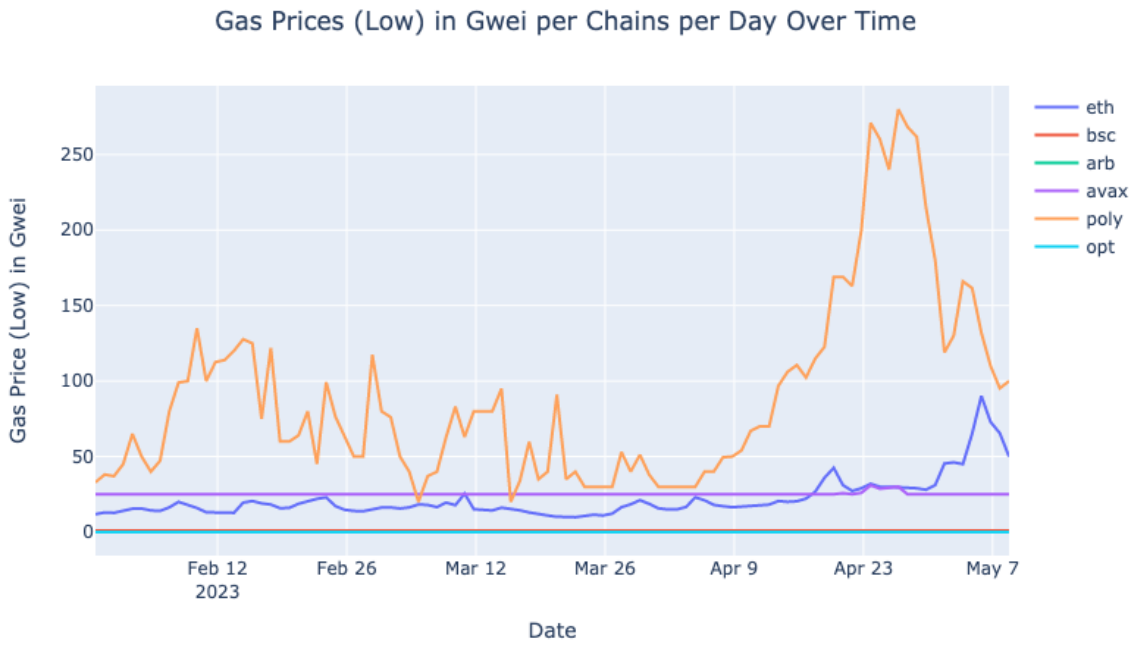


Figure 6.22: Gas Prices in Gwei Over Time per Evaluation Chain

6.2.3.2 Historical Gas and Fiat Price Evaluation

The historical gas and fiat prices of the evaluation chains depicted in Figures 6.22 and 6.23 underscore how the evaluation results of this thesis could vary if the underlying price values were recorded on a different day. The shown high volatility of gas and fiat prices is an important factor to consider when evaluating the results of this thesis.

Figure 6.22 depicts the gas prices in Gwei for multiple evaluation chains. Thereby, it can be observed how especially the gas prices of Polygon and Ethereum vary heavily within the evaluated timespan. For other chains, such as Avalanche, Binance, Arbitrum, or Optimism, the gas prices are more consistent or at least have a much lower variance compared to Polygon or Ethereum.

Having a look at the fiat prices over the same timespan, Figure 6.23 shows how the Ethereum fiat price significantly fluctuates as well. However, for the Polygon layer-2 blockchain, which showed volatile gas prices (*cf.* Figure 6.22), the price in USD is stable over time. This can be seen better in Figure 6.24, where the chains which have Ether as a native currency are excluded. All the prices shown in Figure 6.23 refer to one default unit of the corresponding chain. For instance, the price of one MATIC on Polygon, one Ether on Ethereum, or one BNB on Binance. Moreover, Figure 6.22 highlights how the Arbitrum and Optimism chains also use Ether as a native cryptocurrency. Consequently, the lines of the prices for Ethereum, Arbitrum, and Optimism overlap in Figure 6.22 and cannot be distinguished. Eventually, the big price differences which have been shown before concerning costs in USD between the different evaluation chains are emphasized again.

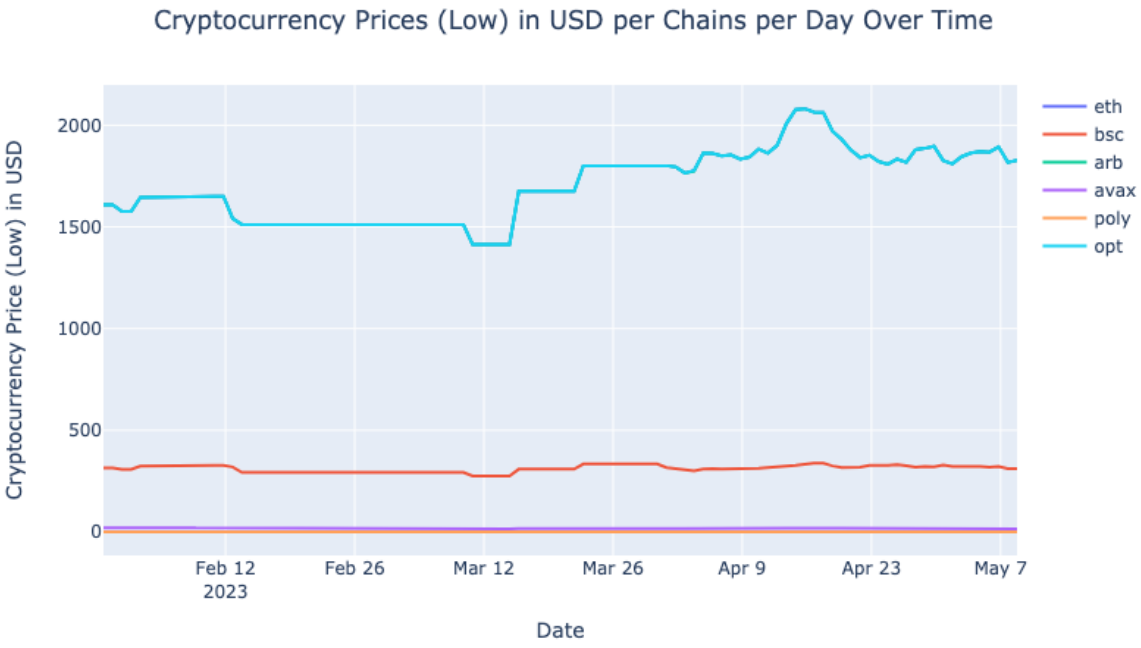


Figure 6.23: Fiat Prices in USD of Evaluation Chains Over Time

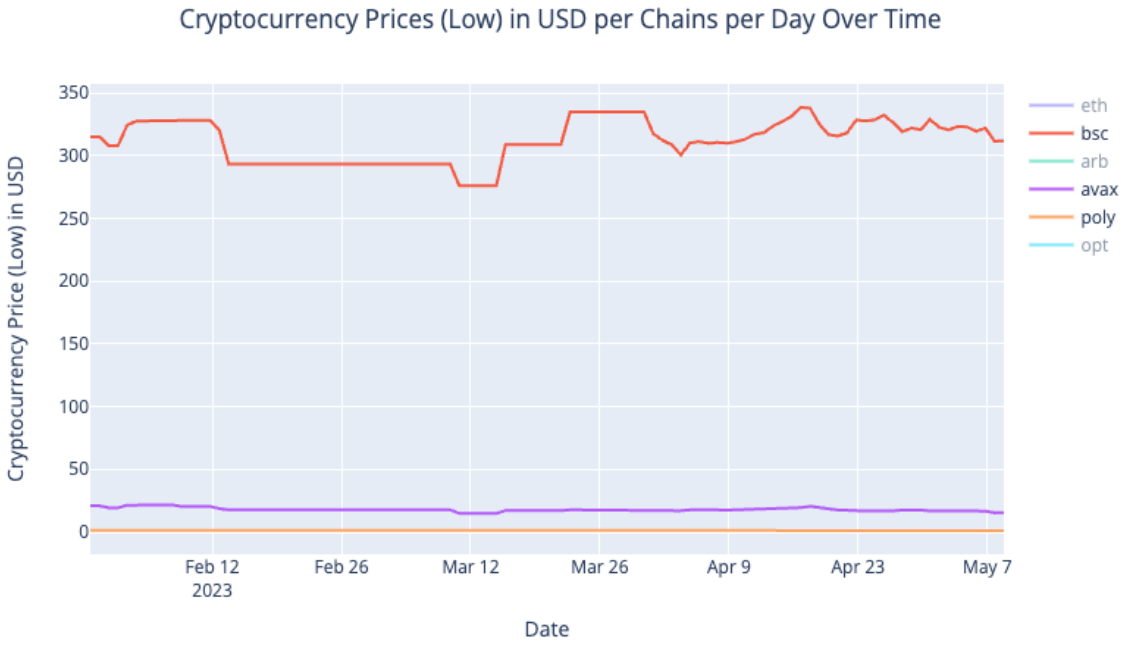


Figure 6.24: Filtered Fiat Prices in USD of Evaluation Chains Over Time

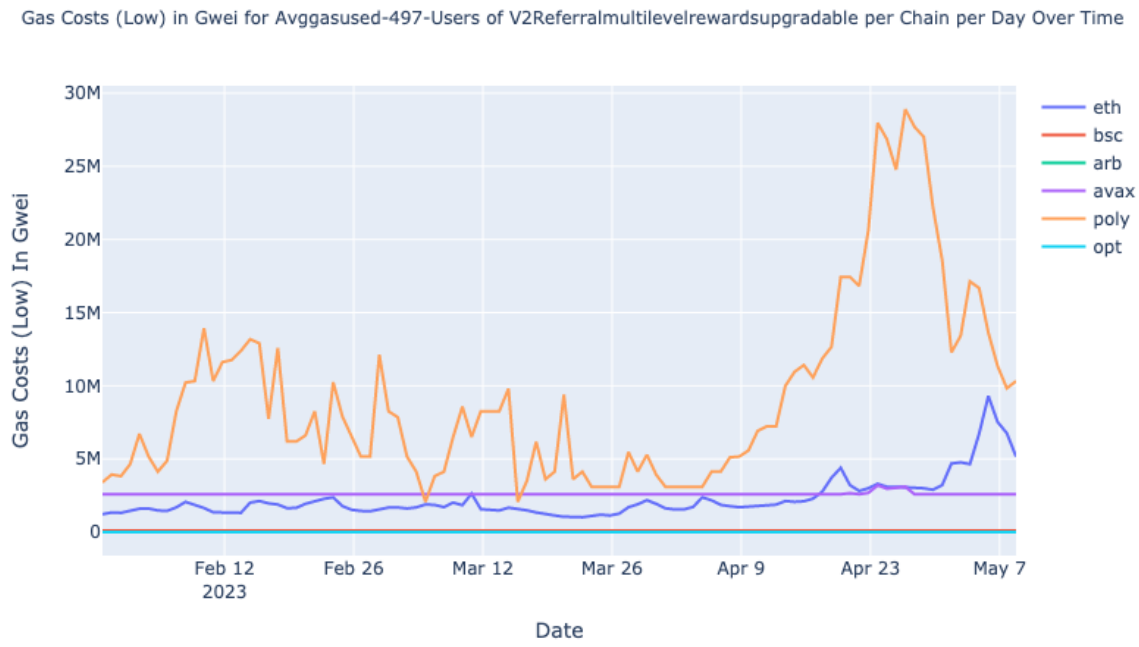


Figure 6.25: Historic Avg Gas Costs in Gwei Over Time for V2ReferralMultilevelRewardsUpgradable

So far, all evaluation results which are related or calculated by using either the gas price of a specific evaluation chain or its cryptocurrency fiat price demonstrate only a snapshot. This snapshot is based on the gas and fiat price values that were fetched and recorded during the execution of the evaluation scripts. Therefore, the previous section gave an overview of how the external and varying values of gas, as well as fiat prices, developed over time. Based on the values shown in Figures 6.22 and 6.23, the gas and fiat costs of the different solution contracts can be recalculated to display the transaction costs for any Deferral solution contract in the form of gas and fiat costs over time.(*cf.* Section 6.2.1.4).

At this point, the results for the historic gas and fiat costs for version two of the multilevel referral rewards contract are displayed and discussed. More results for a few selected contracts can be found in the Appendix in Section D.7.4.

Figure 6.25 displays the average transaction gas costs in Gwei for the evaluation run executed with 500 users of version two of the multilevel rewards contracts. Thereby, the average gas used metric of 103 211 is multiplied by the historic gas prices displayed in Figure 6.22. Eventually, the results in these two figures show the same pattern.

Correspondingly, in Figure 6.26 the historic average transaction costs in USD for the V2ReferralMultilevelRewardsUpgradable contract evaluation run executed with 500 users can be observed. Table D.15 portrays how the average transaction costs on Ethereum is almost 17 USD. As mentioned before, the evaluation result data was generated on the 5th of May 2023. A closer inspection of the Ethereum costs in Figure 6.26 confirms this insight around this date. This highlights how the eventual transaction costs concerning gas and fiat prices for the evaluation results of this thesis could vary. For instance, if the evaluation had been conducted two weeks earlier, the transaction costs in USD on

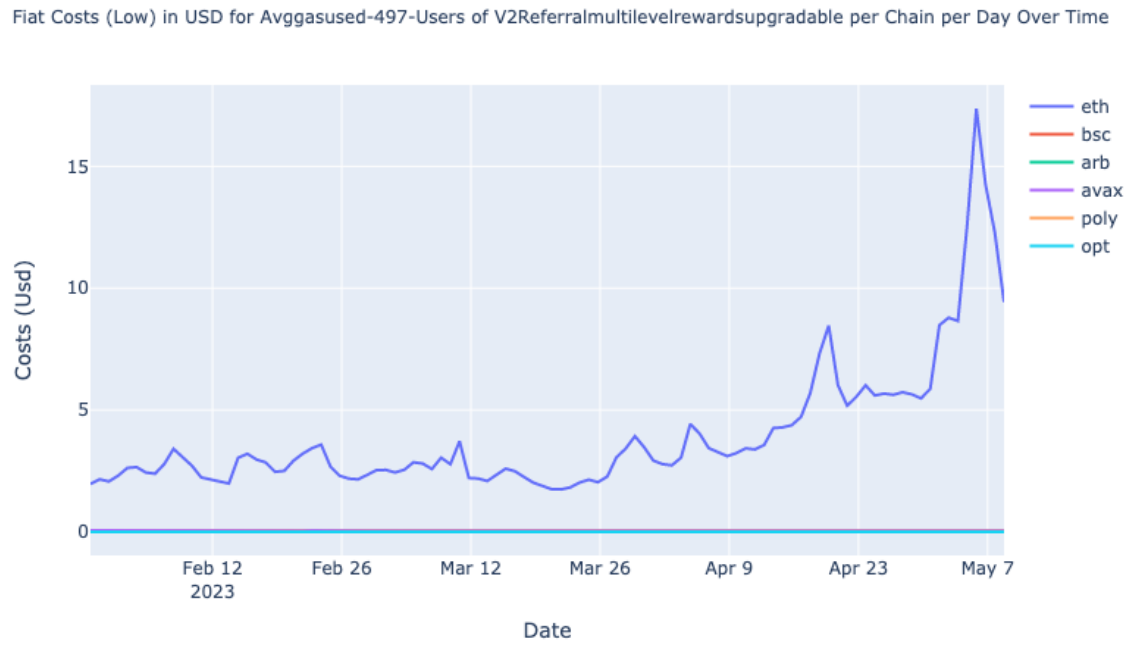


Figure 6.26: Historic Avg Costs in USD Over Time for V2ReferralMultilevelRewardsUpgradable

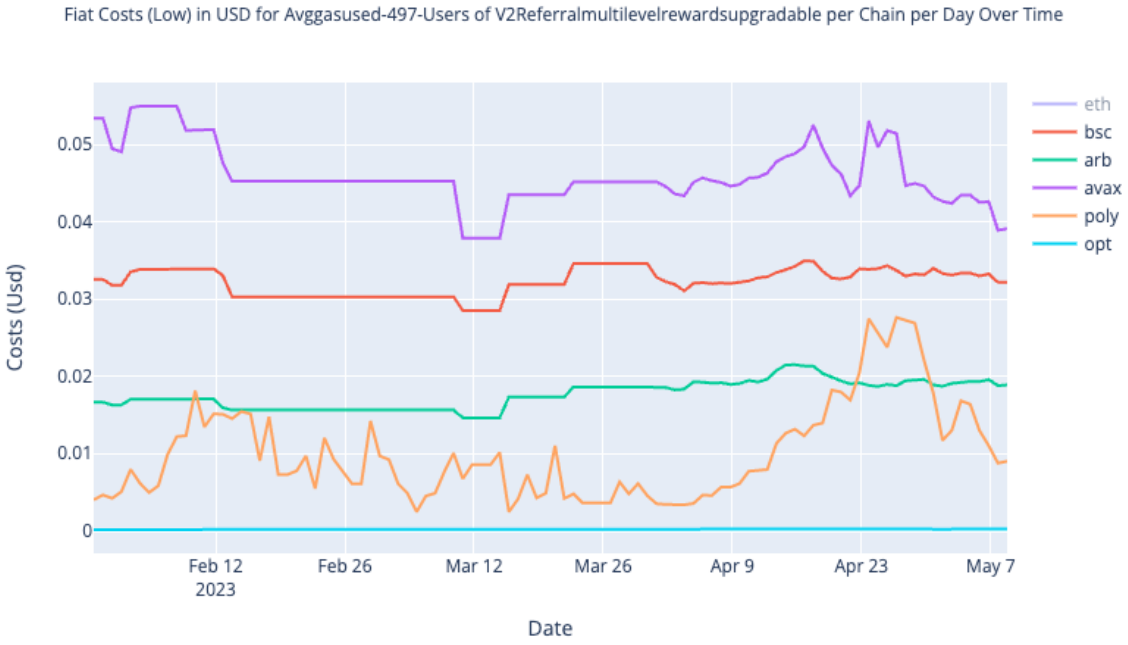


Figure 6.27: Filtered Historic Avg Costs in USD Over Time for V2ReferralMultilevelRewardsUpgradable

Ethereum would have been only half as much, as illustrated in Figure 6.26.

Eventually, for better visibility, Figure 6.27 displays the same results again without the Ethereum costs in USD. Thereby, the different average transaction costs over time for all other evaluation chains can be inspected. Again, it shows how strongly the costs could have changed over time, but this time at a much smaller scale in terms of costs in USD. While the costs of the Polygon chain also fluctuate heavily, other evaluation chains show more consistent costs over time. Therefore, if the evaluation had been performed on a different date, the average transaction costs in USD on the Binance Smart Chain, for instance, would not have exhibited many different values.

After all, Section D.7.4 in the Appendix shows comparable outcomes for further selected Deferral solution contracts. Nonetheless, in general, it emerges how these results are mainly impacted by two factors. First, the cost value in terms of gas used is used as a baseline for the gas and fiat costs calculations. Second, influences come from the fluctuating external metrics, which are collected for the gas prices and cryptocurrency *i.e.,* fiat prices on various blockchains.

6.3 Discussion

In the previous sections, the results of the different Deferral solution contracts' evaluations have been displayed and analyzed. Thereby, different key factors, such as the costs in terms of gas used or the performance with regard to the referral transaction duration, have been examined for the individual smart contract solutions. Especially the metrics of gas used for the referral payment transactions revealed various insights about the different solutions, their design, and implementation. However, the results that could be observed in terms of transaction duration were not as clear. Nonetheless, in general, they supported the overall observations made during the evaluations of the gas used results.

Furthermore, the overall transaction costs concerning gas used, gas costs, as well as fiat costs in USD have been evaluated across all solution contracts for the different evaluation blockchains. Moreover, historic gas and cryptocurrency *i.e.,* fiat price data, have been taken into consideration to examine the resulting costs of the different contracts over a certain time span. Building on these results and findings, the big picture of the Deferral solution can be analyzed.

6.3.1 Solution Requirements

Back in Section 4.2, three requirements that must be met and fulfilled by any solution referral system for this thesis have been introduced. First, a solution should be able to track and evaluate multiple referral processes of different users and evaluate these according to the defined referral conditions. Therein the eligible users and the amount of rewards they receive should be determined.

Ultimately, all presented Deferral smart contracts fulfill this requirement. As has been

outlined in this thesis, the various solution designs have different mechanisms to define referral conditions and to examine and evaluate the referral processes (*cf.* Sections 4.4 and 5.2). In addition, all but one solution contract allow the referral conditions to be updated without interrupting or disrupting the ongoing referral processes. Changing the referral conditions on the go could prove functional for companies that launched and operate the referral system but could, at the same time, have negative effects on the user experience. However, updating the referral conditions could be easily prevented with minor adjustments to the presented solution implementations. The two main features or measures for the evaluation of the referral processes have been shown to be the accumulated value of referral payments and their quantity. Thus, for most of the solution contracts, thresholds for either one or both of these values must be surpassed by referees to become eligible for rewards. In most cases, the amount of the rewards is then defined as a percentage of the total value of all payments. Users can become eligible if they are referrers or if they are the referees who completed the process. Moreover, prior referrers can become eligible for rewards if the referral system *i.e.,* smart contract, has implemented multilevel reward functionalities.

Next, solution contracts must be able to distribute the rewards to eligible users who have been determined by the evaluation processes. Considering this requirement, all Deferral solutions are able to distribute the rewards to eligible users. Thereby the different solutions have introduced the possibility of specifying one or two-sided reward allocations. Rewards can either be distributed to the referee or the referrer only or to both sides. Therein, the proportion of the rewards can be set arbitrarily. Beyond that, several solution contracts have implemented the functionality to distribute rewards along multiple levels. Eventually, it has shown that depending on how many levels and to how many users rewards are distributed, the costs of either evaluating the process or distributing the rewards can be higher.

After all, the last crucial solution requirement for all Deferral solution contracts is the robustness and security of its design and implementation in a decentralized environment. Therefore, all solution contracts are based on the referral payment concept (*cf.* Section 4.3) and can be seen as a referral cash-back program or system. By only accepting referral payment transactions where users must send cryptocurrency assets, the system designs incorporate financial disincentives for potentially malicious users trying to exploit a system. An additional design decision that has been introduced for the multilevel rewards referral evaluators requires referring users to be registered customers *i.e.,* have made previous referral payments themselves. This measure increases the barrier for users to use another account or address controlled by themselves and thereby further preserves the basic idea of a referral system in a decentralized environment. Lastly, the different smart contract implementations have been well tested, ensuring the security and robustness of the implementations to a certain degree (*cf.* Section 6.1)

At large, it can be noted that the Deferral solution contracts fulfill all the solution requirements to be considered a decentralized referral system in the current context. Nevertheless, the main goal of this thesis includes conducting and evaluating the feasibility of a decentralized referral system dealing with high volumes (*cf.* Section 1.2). To do so, the applicability of the presented solutions must be assessed and discussed in the context of real-world scenarios.

6.3.2 Feasibility and Real World Applicability

By fulfilling all the defined solution requirements, the different referral systems have already contributed to their feasibility from a technical point of view. Yet, aside from the technical design and implementation, other factors, such as the conceptual design, play important roles in the feasibility and success of referral systems (*cf.* Sections 3.1 and 4.2).

In a real-life use case, a Deferral smart contract would need to be integrated into a functioning payment process flow of a company. Thus, in order for a company to deploy and operate a Deferral referral system, it must accept payments for its products or services in the form of cryptocurrencies or ERC20 tokens. This is crucial as the referral payments are the foundation of the Deferral solution architecture. For the deployment of the referral contract to a compatible blockchain, the company would represent the receiver address to which the referral payments are forwarded to. All users *i.e.,* customers of this company could then participate in the referral system or program when they make purchases *i.e.,* payments for a product or service they have bought. By participating in the referral program, the customers could earn a cashback on their payments in the form of referral rewards.

Apart from accepting cryptocurrency payments, the company would also have to offer a corresponding user interface that enables access to the referral system. Through this user interface, customers would be able to send their referral payments to the Deferral contract and include the required information *i.e.,* the referrer address. For any company that already has the required cryptocurrency payment processes and flows in place, integrating a Deferral solution contract would not come with a lot of additional costs and effort. The user interface on the customer side could be kept straightforward. Nonetheless, to ensure a good user experience, a potential user interface should consider incorporating short and clear feedback cycles about the progress of any user and overall a good explanation of the general referral conditions (*cf.* Chapter 3). Especially since, as outlined in Section 3.1, the user experience can be a crucial factor for the success of a referral system.

6.3.2.1 Economic Design

Apart from the technical infrastructure, including suitable payment processes and accessible user interfaces, the economic design of a referral system *i.e.,* its conditions and rewards, is crucial for the feasibility as well (*cf.* Sections 3.1.1 and 3.1.2). Thereby, it is important to define suitable rewards and practicable referral conditions.

For an exemplary company that wants to implement a Deferral decentralized referral system, several steps would need to be considered. First of all, it must be assessed what kind of referral payment users would perform. For instance, users might usually pay small amounts to the company but frequently. As a result, it would not be optimal for the company to choose a Deferral Solution with higher average gas costs per transaction, like the V2ReferralMultilevelRewardsUpgradable solution. Moreover, if the referral conditions were set to have a high payment value or payment quantity thresholds, participation in the referral system would become less lucrative for the users.

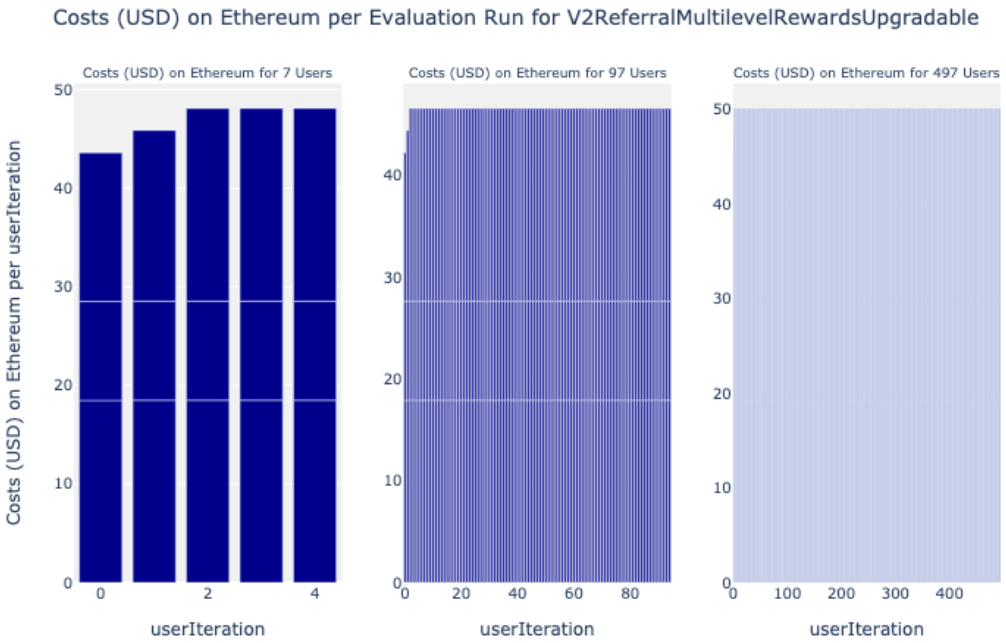


Figure 6.28: Ethereum Transaction Costs in USD per UserIteration for V2ReferralMultilevelRewardsUpgradable

To get a better overview, an exemplary scenario including company (A) can be assumed. Company (A) offers a subscription service for which users pay 20 USD on a monthly basis. The 20 USD can be paid in cryptocurrencies. Now, company (A) wants to implement a referral system to boost its customer engagement and potentially reach new customers. To do so, they decide to deploy the V2ReferralMultilevelRewardsUpgradable contract to the Ethereum chain. Based on the results and the collected gas and fiat prices, the deployment of the contract could cost the company around 200 USD. The referral conditions of the contract would be set in a similar way to the V2ReferralMultilevelRewardsUpgradable evaluation configuration. Every user must make at least two payments for the subscription of a total value of 20 USD or more. The referral reward, as well as the referee reward, would be set to 50%, and the maximum reward level would be set to three. Consequently, users would complete the referral process with the second subscription payment. Thereon, the user and their referrers would each receive a cashback of 10 USD of the total 20 USD reward.

Considering the gas and USD prices for Ether which have been used as a base for the calculation of the results in this context, the costs per user shown in Figure 6.28 highlight how this specific setup of the referral system and the rewards would be pointless for the users *i.e.,* customers of company (A). Since all the referral payment transaction costs are paid by the user, it would cost the customers of the company (A) almost 50 USD to execute the required referral payment transactions and complete the process. Compared to the potential cashback referral reward of 20 USD, participation in the referral system would not be profitable. Thus, customers would decide not to participate in the referral system and execute regular payment transactions, which would be much cheaper in this

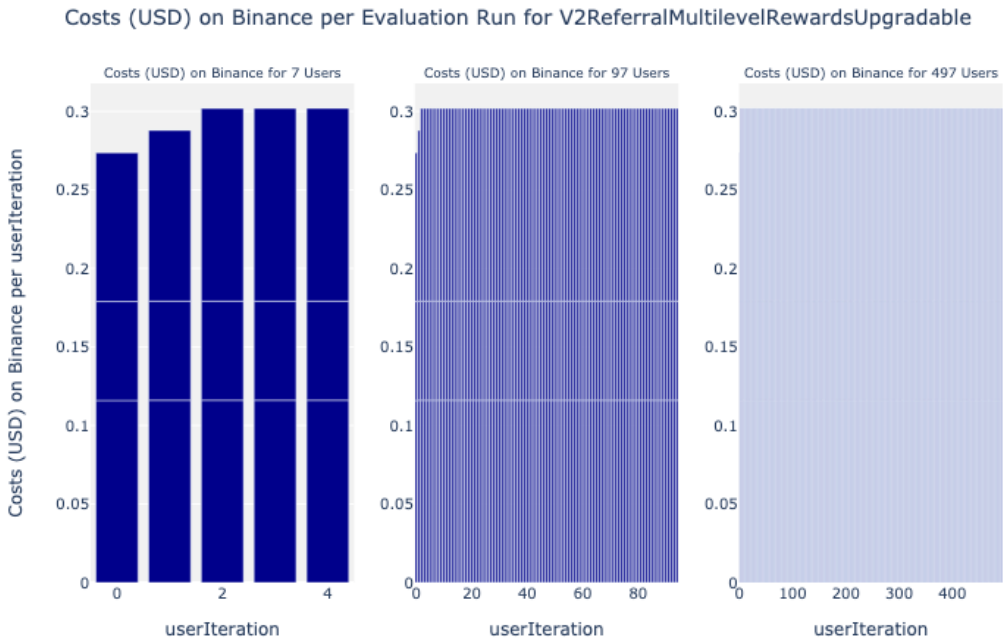


Figure 6.29: Binance Transaction Costs in USD per UserIteration for V2ReferralMultilevelRewardsUpgradable

case.

However, company (A) choosing a different blockchain to deploy and operate the referral smart contract in the same scenario would reveal a different picture. By using the Binance Smart Chain, the costs for the users to complete the referral process would already have gone down to less than one USD (*cf.* Figure 6.29). In comparison to the unchanged referral reward of 20 USD, the same referral system on a different chain would become lucrative for users. By choosing even cheaper blockchain alternatives *e.g.,* Polygon, Arbitrum, or Optimisms, the costs could be reduced even more. Eventually, this would also bring new wiggle room for company (A) to adjust the referral rewards or conditions. For instance, the participation would remain lucrative for customers even with a total referral reward percentage of 20%. Nonetheless, by choosing less common or adopted chains, potential users might be discouraged from participating.

This example portrays the challenges and trade-offs that would need to be considered when implementing and deploying a Deferral solution contract in a real-world scenario. On the one hand, it shows how the underlying technical implementation *i.e.,* the technical design, can influence the conceptual design *i.e.,* referral rewards and conditions, and vice versa. On the other hand, it exemplifies how the design, implementation, and operation of a decentralized referral system also depend on the real-world environment and circumstances relevant to the provider of the system *e.g.,* company (A). For instance, in the example scenario above, if company (A) would sell luxury products where users would pay larger amounts within every payment transaction *e.g.,* around 1 000 USD, the potentially higher referral reward could also exceed the in comparison high costs for the referral payment transactions on the Ethereum chain.

Eventually, the economic design of a decentralized referral system in the current context would always lead to a game of ping-pong where the appropriate setup and configuration have to be found. Hence, it is always difficult to identify the optimal technical and conceptual or economical design of such a referral system.

Other crucial aspects which further complicate these processes are the strongly changing and fluctuating gas and cryptocurrency *i.e.,* fiat prices, which have been discussed in Sections 6.2.3.1 and 6.2.3.2. Hence, the costs that are associated with a Deferral solution contract and a decentralized referral system can vary significantly in a short period of time for both the users and the system providers *i.e.,* the companies. Against this backdrop, the functionality that allows system providers to update the referral conditions and rewards for most of the presented Deferral solutions could be useful to react and adapt to changing gas and fiat prices. In this way, the lucrativity of the referral system for users could be ensured or at least supported to a certain degree.

6.3.2.2 Business Model

The previous section has provided one example of how a Deferral solution contract could be used by companies. Besides, the Deferral solution and the Solidity contracts could also be instituted as a white-label solution, offering decentralized referral systems as a service.

Thereby, companies that are interested in offering a decentralized referral system could contact the Deferral institution and agree upon the referral conditions, rewards, and the deployment environment *e.g.,* the blockchain. Afterward, the Deferral contract could be deployed with the defined conditions and corresponding receiver address of the company. Users or customers of the company could still be directed to interact with the Deferral contract through a user interface. Hence, the process would look very similar from a user perspective. Additionally, with regards to the business model of the Deferral institution, it would be straightforward to adopt the implementation of the Solidity smart contracts and collect a small percentage of the referral payments or the distributed rewards as a source of income. Nonetheless, further exploration in this direction is reserved for future research.

6.3.3 Decentralization

Ultimately, it is worth reviewing the big picture of decentralized referral systems in the context of the developed Deferral solution. Based on the presented results, it can be discussed which components of the design and implementation of a decentralized referral system are best suited for decentralization and what aspects could benefit from centralization.

As demonstrated in the evaluation section, the evaluation of the referral processes *i.e.,* the storage of process-relevant data for each user on the smart contracts, has shown to be costly in terms of gas used. Thus, storing and or evaluating the referral processes off-chain could be considered to reduce the costs.

The Attrace [18] example, as discussed in Section 3.4.1.1, employs a similar approach by using a blockchain oracle to track and evaluate data relevant to the referral process evaluation and the eventual reward definition. Ideally, a decentralized oracle could be incorporated to avoid introducing centralized components. However, a centralized oracle can also be employed, as demonstrated by the Attrace example. Eventually, implementing the evaluation of processes and determining eligible users and their rewards off-chain could help reduce costs.

Regarding reward distribution, providing an off-chain alternative would be more challenging and not necessarily advantageous, particularly when rewards come in the form of cryptocurrencies or ERC20 tokens. However, as the payout of rewards could become increasingly expensive *e.g.,* with deep referral chains for multilevel rewards, it may be necessary to reconsider the on-chain reward distribution approach.

Moreover, it is worthwhile to compare the existing introduced solutions for decentralized referral system approaches, such as Attrace [18] (*cf.* Section 3.4.1.1) and the Energi affiliate program [50] (*cf.* Section 3.4.1.2), with the Deferral solution. Therein, both Attrace and Energi incorporate the principle of referral payments, where the transactions involve either purchasing cryptocurrencies, in the case of Attrace, or providing liquidity on the Energiswap DEX.

At last, it can be maintained that the Deferral solution and all components of the resulting referral system can remain fully decentralized as they are implemented on a smart contract that can be deployed to a blockchain. Nonetheless, there is one notable exception. As previously discussed (*cf.* Section 4.4.2.1), the use of upgradable contracts [97] introduces a certain degree of centralization to the final solution, slightly diminishing the final level of decentralization. However, all the solution contracts could also be implemented without the upgradable functionality.

Chapter 7

Final Considerations

7.1 Summary

Within this thesis, the subject of decentralized referral systems has been researched, introducing related work covering essential components of the design and implementation of referral systems and programs. It has been shown which components concerning the conceptual and technical design must be considered when developing and implementing a decentralized referral system. Existing solutions and approaches for centralized and decentralized referral systems have been portrayed.

Furthermore, the solution architecture and design of the Deferral solution have been conceptualized as a base for implementing decentralized referral systems in this context. The developed solution comprises several solution designs and has implemented multiple Solidity smart contracts that can represent the technical realization of a decentralized referral system on a blockchain. The Deferral solution prototypes' security, costs, and performance have been tested and evaluated. Implications of high volumes of participating users and fluctuating gas, as well as cryptocurrency *i.e.,* fiat prices, on the transaction costs, have been examined.

All relevant details have been documented and outlined in this thesis and in the two corresponding GitHub repositories [132, 133]. Following the testing and evaluation of the Deferral smart contracts, the results of the contract evaluations have been displayed, analyzed, and discussed. At the same time, the generated results have been compared across different suitable blockchains to establish benchmarks for gas and fiat costs calculated based on the underlying gas and cryptocurrency prices. The advantages and disadvantages of the different solution designs and their implementations have been outlined. Additionally, the evaluation results have been embedded into historical data illustrating the varying gas as well as fiat prices on different blockchains over time.

Thereafter, the feasibility of a decentralized referral system with a focus on technical implementation and the defined solution requirements has been discussed. Moreover, resting on the achieved results, the real-world practicability of a decentralized system oriented at the Deferral solution has been debated. The consequential challenges and

issues caused by the volatile nature of gas and cryptocurrency prices have been exemplified. Therein, it showed how these influencing factors would affect the costs of a system and, eventually, the design decisions concerning both its technical implementation as well as the related conceptual or economic design choices. Finally, the parallels between the introduced existing solution approaches and the Deferral solution have been drawn, and the resultant degree of decentralization has been reviewed.

7.2 Conclusion

In conclusion, the feasibility of a decentralized referral system can be confirmed within the scope of this thesis. The multiple Deferral smart contract solutions present well-tested and documented prototypes for the technical implementation and can be deployed decentrally to different blockchains. Thereon, the evaluation process of this thesis has uncovered several learnings and findings.

First, the analysis showed that the costs of referral transactions rise the more data is stored and the more functionality is implemented on the evaluator smart contracts. Thereby, it became clear how the initial storage of data is a main driving factor for costs. The distribution of the rewards could be identified as the second large cost driver. Consequently, it became evident that the start, as well as the completion of a decentralized referral process, are associated with more costs than regular referral transactions that would only update the data. Thus, depending on the contract implementation and the number of users receiving rewards upon the completion of a referral process, either the starting or the completing transaction is the most expensive for users.

Second, the evaluation demonstrated that for all but one solution smart contract, their design is not negatively affected by high volumes of participating users concerning transaction costs. Thus, with the exception of the V1ReferralMultilevelRewardsUpgradable contract, all solution contracts also represent a feasible referral system when dealing with high volumes of users.

Furthermore, the results and evaluation of the gas and cryptocurrency *i.e.,* fiat prices, revealed the different cost structures depending on the chosen blockchain. This investigation established how the volatility of these external prices, which are used to calculate the transaction costs, can influence the resulting costs of a decentralized referral system. Further, it became clear that the gas costs of a transaction can often be misinterpreted, primarily due to inconsistencies in their units of measurement. In general, these findings underlined that transaction cost results that are tied to gas and or fiat prices must be interpreted with caution. Eventually, the incorporation of historical price data depicted how fast the costs associated with decentralized referral system transactions can change.

Eventually, the discussion of the results established how complicated and multilayered the design and implementation of a decentralized referral system can be. It was seen that various environmental factors of a prospective company could influence the decision related to the operation of such a system.

The comparison of the Deferral solution to existing introduced approaches for decentralized referral systems revealed several commonalities. Finally, the results pointed out what components of the design and implementation could be established off-chain for further improving the costs and performance of a decentralized referral system.

7.3 Limitations and Future Work

While the solution of this thesis aims to present a tested and documented framework for the implementation of decentralized blockchain-based referral systems, it is important to acknowledge the limitations that arose during the course of the research and implementation. Firstly, not all existing projects or related solutions for decentralized referral systems in the fast-moving field of decentralized applications were covered and researched within the scope of this thesis. Other areas or programs related to referral systems, such as loyalty programs, could also entail helpful and relevant conceptual or technical aspects and were not considered during this thesis. Secondly, the implemented solution smart contracts were developed in Solidity. Hence they are only applicable to EVM-based and Solidity-compatible blockchains. There are several limitations to the tests and evaluations of the solution contracts. The smart contracts have not explicitly been tested on mainnet blockchains due to the involved costs. The defined evaluation blockchains were used as a reference point for transaction costs and not to test and execute the smart contracts. Moreover, the impact of high volumes of data and transactions has been tested concerning the number of users participating in the system. Thereby, two limitations could be observed. On the one hand, no reference value for high volumes of participating users in a decentralized referral system was considered during the evaluation phase. On the other hand, important performance aspects such as the throughput, transaction frequency, or block time of different blockchain networks have not been considered. Lastly, the implemented solutions are prototypes and do not encompass all components required to operate a referral system in a real-world environment. As a result, the Deferral solutions were not tested with real users.

The presented results of this thesis and their limitations open up several possible paths for future work. To begin with, the presented solution designs and implementations could be extended concerning the referral process evaluation. Additional referral conditions *e.g.,* a time constraint on the ongoing referral processes for the different contracts, could be explored. Thereby, the architecture and designs of the smart contracts could be transposed to incorporate off-chain data or functionality to further optimize costs. Next, it would be interesting to explore and test the implemented smart contracts on other blockchain platforms and ecosystems, such as Solana or Polkadot. Thereby, additional blockchains could be included in the evaluation process. Moreover, performance metrics, such as the transaction throughput of different blockchain networks, could be analyzed to evaluate the impact of high volumes on the solution prototypes. Additionally, the results of this thesis could be re-evaluated with higher numbers of participating users in future work. The solution smart contracts could also be tested and applied in real-world use cases on mainnet blockchains. Therein, a user interface for the Deferral contracts could be implemented and evaluated with real users. After all, the business model of a potential

whitelabel Deferral solution, which offers decentralized referral systems as a service, could be further investigated.

Bibliography

[1] Adam Rapp and Lisa Beeler. "The state of selling & sales management research: a review and future research agenda". In: *Journal of Marketing Theory and Practice* 29.1 (2021), pp. 37–50.

[2] John Adler et al. "Astraea: A Decentralized Blockchain Oracle". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pp. 1145–1152.

[3] Shubhani Aggarwal and Neeraj Kumar. "Chapter Seven - Basics of blockchain". In: *The Blockchain Technology for Secure and Smart Applications across Industry Verticals*. Ed. by Shubhani Aggarwal, Neeraj Kumar, and Pethuru Raj. Vol. 121. Advances in Computers. 2021, pp. 129–146.

[4] Hamda Al-Breiki et al. "Trustworthy Blockchain Oracles: Review, Comparison, and Open Research Challenges". In: *IEEE Access* 8 (2020), pp. 85675–85685.

[5] Maher Alharby and Aad van Moorsel. "Blockchain-based Smart Contracts: A Systematic Mapping Study". In: *CoRR* abs/1710.06372 (2017).

[6] Amazon. *Amazon*. [Online] `https://www.amazon.com`, last visit January 2023.

[7] Ambassador. *Reduce acquisition costs and increase customer value with Ambassador*. [Online] `https://www.getambassador.com`, last visit January 2023.

[8] American Marketing Association. *What is Digital Marketing?* [Online] `https://www.ama.org/pages/what-is-digital-marketing/`, last visit December 2022.

[9] Andrew Reed, *What Is Referral Fraud in Ecommerce? (And What You Can Do About It)*. [Online] `https://www.clean.io/blog/what-is-referral-fraud-in-ecommerce-and-what-you-can-do-about-it`, last visit January 2023.

[10] Alexander Anter. *What happened to RefToken*. [Online] `https://reftoken.io/blog/`, last visit January 2023.

[11] Johan Arndt. "Role of Product-Related Conversations in the Diffusion of a New Product". In: *Journal of Marketing Research* 4.3 (1967), pp. 291–295.

[12] Attrace. *Attrace Protocol*. [Online] `https://attrace.com/about`, last visit January 2023.

[13] Attrace. *Attrace Roadmap*. [Online] `https://attrace.com/about/roadmap`, last visit January 2023.

[14] Attrace. *Attrace Token*. [Online] `https://attrace.com/about/attrace-token`, last visit January 2023.

[15] Attrace. *Oracle Staking*. [Online] `https://attrace.com/about/tokenomics`, last visit January 2023.

[16] Attrace. *Oracles*. [Online] https://attrace.com/about/oracles, last visit January 2023.

[17] Attrace. *Overview Attrace Referral Farms*. [Online] https://medium.com/attrace/overview-attrace-referral-farms-52b2f88f05af, last visit January 2023.

[18] Attrace. *Referral Protocol for Web3*. [Online] https://attrace.com/, last visit January 2023.

[19] Rodrigo Belo and Ting Li. "Social Referral Programs for Freemium Platforms". In: *Management Science* (Apr. 2022).

[20] Abdeljalil Beniiche. "A Study of Blockchain Oracles". In: *CoRR* abs/2004.07140 (2020).

[21] Barry Berman. "Referral marketing: Harnessing the power of your customers". In: *Business Horizons* 59.1 (2016), pp. 19–28.

[22] Binance. *BNB Chain*. [Online] https://www.bnbchain.org/en, last visit May 2023.

[23] George Bissias et al. "Sybil-Resistant Mixing for Bitcoin". In: *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. WPES '14. 2014, 149–158.

[24] BitTorrent. *BitTorrent*. [Online] https://www.bittorrent.com/, last visit January 2023.

[25] Eyal Biyalogorsky, Eitan Gerstner, and Barak Libai. "Customer Referral Management: Optimal Reward Programs". In: *Marketing Science* 20.1 (2001), pp. 82–95.

[26] Jacques Bughin, Jonathan Doogan, and Ole Jorgen Vetvik. "A new way to measure word-of-mouth marketing". In: *McKinsey Quarterly* 2.1 (2010), pp. 113–116.

[27] Vitalik Buterin et al. "A next-generation smart contract and decentralized application platform". In: *white paper* 3.37 (2014), pp. 2–1.

[28] Francis A. Buttle. "Word of mouth: understanding and managing referral marketing". In: *Journal of Strategic Marketing* 6.3 (1998), pp. 241–254.

[29] Wei Cai et al. "Decentralized Applications: The Blockchain-Empowered Software System". In: *IEEE Access* 6 (2018), pp. 53019–53033.

[30] Giulio Caldarelli. "Understanding the Blockchain Oracle Problem: A Call for Action". In: *Information* 11.11 (2020).

[31] Giulio Caldarelli and Joshua Ellul. "The Blockchain Oracle Problem in Decentralized Finance—A Multivocal Approach". In: *Applied Sciences* 11.16 (2021).

[32] cgewecke - Christopher Gewecke. *Hardhat Gas Reporter Plugin*. [Online] https://github.com/cgewecke/hardhat-gas-reporter#readme, last visit April 2023.

[33] Chainlink. *Securely connect smart contracts with off-chain data and services*. [Online] https://chain.link, last visit January 2023.

[34] Yan Chen and Cristiano Bellavitis. "Blockchain disruption and decentralized finance: The rise of decentralized business models". In: *Journal of Business Venturing Insights* 13 (2020), e00151.

[35] Coinbase. *Buy & Sell Bitcoin, Ethereum, and more with trust*. [Online] https://coinbase.com, last visit January 2023.

[36] Coinbase. *The Coinbase referral program*. [Online] https://help.coinbase.com/en/coinbase/other-topics/other/the-coinbase-referral-program, last visit January 2023.

[37] Coinbase. *What is Coinbase?* [Online] `https://help.coinbase.com/en/coinbase/getting-started/crypto-education/what-is-coinbase`, last visit January 2023.

[38] CoinGecko. *CoinGecko API Documentation.* [Online] `https://www.coingecko.com/de/api/documentation`, last visit April 2023.

[39] Jens Cornelsen and Hermann Diller. "References within the context of customer valuation". In: *IMP Conference (14th)*. Vol. 14. IMP. 1998.

[40] George Coulouris et al. *Distributed Systems: Concepts and Design.* 5th. 2011.

[41] Chris Dannen. *Introducing Ethereum and solidity.* Vol. 1. 2017.

[42] Massimo Di Pierro. "What Is the Blockchain?" In: *Computing in Science Engineering* 19.5 (2017), pp. 92–95.

[43] John R. Douceur. "The Sybil Attack". In: *Peer-to-Peer Systems.* Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. 2002, pp. 251–260.

[44] Daniel Drescher. *Blockchain Basics : A Non-Technical Introduction in 25 Steps.* 2017.

[45] Dropbox. *Dropbox Website.* [Online] `https://www.dropbox.com/`, last visit January 2023.

[46] Dropbox. *How to refer friends to Dropbox and get more storage space.* [Online] `https://help.dropbox.com/storage-space/earn-space-referring-friends`, last visit January 2023.

[47] Energi. *Energi GitHub.* [Online] `https://github.com/energicryptocurrency`, last visit January 2023.

[48] Energi. *Energi Whitepaper - The Safest Blockchain in the World.* [Online] `https://eadn-wc01-5393995.nxedge.io/wp-content/uploads/2022/02/Energi-White-Paper-February-2022.pdf`, last visit January 2023. 2022.

[49] Energi. *Energiswap - Decentralized Trading Powered by Energi.* [Online] `https://energiswap.exchange/`, last visit January 2023.

[50] Energi. *Energiswap Referral Program — the First Truly Decentralized Affiliate Reward Offering in Crypto.* [Online] `https://medium.com/energi/energiswap-referral-program-the-first-truly-decentralized-affiliate-reward-offering-in-crypto-7e23437f13c7`, last visit January 2023.

[51] Energi. *The Safest Blockchain for Users Businesses You Everyone.* [Online] `https://energi.world/`, last visit January 2023.

[52] ESLint. *Find and fix problems in your JavaScript code.* [Online] `https://eslint.org/`, last visit April 2023.

[53] Ethereum. *Decentralized Autonomous Organizations (DAOs).* [Online] `https://ethereum.org/en/dao/`, last visit January 2023.

[54] Ethereum. *Introduction to Smart Contracts.* [Online] `https://ethereum.org/en/developers/docs/smart-contracts/`, last visit January 2023.

[55] Ethers. *Ethers Documentation.* [Online] `https://docs.ethers.org/v5/`, last visit April 2023.

[56] Ankit Gangwal, Haripriya Ravali Gangavalli, and Apoorva Thirupathi. "A survey of Layer-two blockchain protocols". In: *Journal of Network and Computer Applications* 209 (2023), p. 103539.

[57] GitHub. *GitHub Actions - Automate your workflow from idea to production.* [Online] `https://github.com/features/actions`, last visit April 2023.

[58] Julija Golosova and Andrejs Romanovs. "The Advantages and Disadvantages of the Blockchain Technology". In: *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. 2018, pp. 1–6.

[59] Google. *Google About*. [Online] `https://about.google/`, last visit January 2023.

[60] Zhiling Guo. "Optimal decision making for online referral marketing". In: *Decision Support Systems* 52.2 (2012), pp. 373–383.

[61] Hardhat. *Hardhat Documentation*. [Online] `https://hardhat.org/`, last visit April 2023.

[62] Hardhat. *Testing Contracts*. [Online] `https://hardhat.org/tutorial/testing-contracts`, last visit April 2023.

[63] Sabrina Helm. "Calculating the value of customers' referrals". In: *Managing Service Quality* 13 (Apr. 2003), pp. 124–133.

[64] Andreas Herrmann and Ralph Fürderer. "The Value of Passenger Car Customers". In: *Customer Retention in the Automotive Industry: Quality, Satisfaction and Loyalty*. Ed. by Michael D. Johnson et al. 1997, pp. 349–370.

[65] Kevin Hong et al. "On the Role of Fairness and Social Distance in Designing Effective Social Referral Systems". In: *MIS Quarterly* 41 (Mar. 2017), pp. 787–809.

[66] IPFS. *IPFS powers the Distributed Web*. [Online] `https://ipfs.tech/`, last visit January 2023.

[67] Elvira Ismagilova et al. *Electronic Word of Mouth (EWOM) in the Marketing Context: A State of the Art Analysis and Future Directions*. 1st. 2017.

[68] Jaehwuen Jung et al. "Impact of Incentive Mechanism in Online Referral Programs: Evidence from Randomized Field Experiments". In: *Journal of Management Information Systems* 38.1 (2021), pp. 59–81.

[69] P.K. Kannan and Hongshuang "Alice" Li. "Digital marketing: A framework, review and research agenda". In: *International Journal of Research in Marketing* 34.1 (2017), pp. 22–45.

[70] Chris Karlof and David Wagner. "Secure routing in wireless sensor networks: attacks and countermeasures". In: *Ad Hoc Networks* 1.2 (2003), pp. 293–315.

[71] Anastasii I. Klimin et al. "Referral program in the context of social capital and digital marketing methods". In: *Proceedings of the International Conference on Digital Technologies in Logistics and Infrastructure (ICDTLI 2019)*. 2019/09, pp. 57–60.

[72] Antonia Köster, Christian Matt, and Thomas Hess. "Does the Source Matter? How Referral Channels and Personal Communication Tools Affect Consumers' Referral Propensity". In: *Hawaii International Conference on System Sciences*. 2017.

[73] Di Kuang, Xiao-Fei Li, and Wen-Wen Bi. "How to Effectively Design Referral Rewards to Increase the Referral Likelihood for Green Products". In: *Sustainability* 13.13 (2021).

[74] KuCoin. *Earn free crypto*. [Online] `https://www.kucoin.com/land/task-center`, last visit January 2023.

[75] KuCoin. *Invite Friends to Earn Stars*. [Online] `https://www.kucoin.com/referral`, last visit January 2023.

[76] KuCoin. *Referral Program - Invite Friends to Earn Stars - FAQ*. [Online] `https://www.kucoin.com/de/support/8426175962393`, last visit January 2023.

[77] V. Kumar, J. Andrew Petersen, and Robert P. Leone. "Driving Profitability by Encouraging Customer Referrals: Who, When, and How". In: *Journal of Marketing* 74.5 (2010), pp. 1–17.

[78] V. Kumar et al. “The Power of CLV: Managing Customer Lifetime Value at IBM”. In: *Marketing Science* 27.4 (2008), pp. 585–599.

[79] Amanda Laine. *Insights from Inside the Infamous PayPal Referral Program.* [Online] `https://growsurf.com/blog/paypal-referral-program`, last visit January 2023.

[80] Monica Law. “Customer referral management: the implications of social networks”. In: *The Service Industries Journal* 28.5 (2008), pp. 669–683.

[81] Linktrust. *The Most Trusted Platform in Performance Marketing.* [Online] `https://linktrust.com`, last visit January 2023.

[82] Viral Loops. *How Revolut grew 150x with referral marketing—a case study.* [Online] `https://viral-loops.com/revolut-referral-marketing-case-study`, last visit January 2023.

[83] Viral Loops. *Our Product. Offering a unique experience tailored to you.* [Online] `https://viral-loops.com/product`, last visit January 2023.

[84] Alexander Anter Manuel Granados and Jan Sammut. *RefToken: A decentralized affiliate marketplace.* [Online] `https://reftoken.io/uploads/RefToken-Yellow-Paper.pdf`, last visit January 2023. 2017.

[85] Lodovica Marchesi et al. “Design Patterns for Gas Optimization in Ethereum”. In: *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. 2020, pp. 9–15.

[86] F.J. Martínez-López and D.L. López. *Advances in Digital Marketing and eCommerce: Second International Conference, 2021.* Springer Proceedings in Business and Economics. 2021.

[87] Apostle Mengoulis. *Dropbox grew 3900% with a simple referral program. Here’s how!* [Online] `https://viral-loops.com/blog/dropbox-grew-3900-simple-referral-program/`, last visit January 2023.

[88] Apostle Mengoulis. *The best 100 referral marketing examples to get you inspired.* [Online] `https://viral-loops.com/blog/100-referral-marketing-examples/`, last visit January 2023.

[89] Meta. *Meta.* [Online] `https://www.meta.com`, last visit January 2023.

[90] Du Mingxiao et al. “A review on consensus algorithm of blockchain”. In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2017, pp. 2567–2572.

[91] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena. “An Overview of Smart Contract and Use Cases in Blockchain Technology”. In: *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2018, pp. 1–4.

[92] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: *Decentralized Business Review* (2008), p. 21260.

[93] Node.js. *Node.js - an open-source, cross-platform JavaScript runtime environment.* [Online] `https://nodejs.org/en`, last visit April 2023.

[94] OpenAI. [Online] `https://chat.openai.com/`, last visit May 2023.

[95] OpenZeppelin. *OpenZeppelin Contracts.* [Online] `https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol`, last visit April 2023.

[96] OpenZeppelin. *OpenZeppelin Contracts*. [Online] https://github.com/OpenZeppelin/openzeppelin-upgrades/blob/master/packages/core/contracts/Initializable.sol, last visit April 2023.

[97] OpenZeppelin. *OpenZeppelin Docs - Upgrades*. [Online] https://docs.openzeppelin.com/upgrades, last visit April 2023.

[98] Owloracle. *Owlracle API documentation*. [Online] https://owlracle.info/docs, last visit April 2023.

[99] PayPal. *Refer a Friend*. [Online] https://www.paypal.com/us/webapps/mpp/invite/terms, last visit January 2023.

[100] PayPal. *Shop. Send. Manage.* [Online] https://www.paypal.com/us/home, last visit January 2023.

[101] PayPal. *What is PayPal?* [Online] https://www.paypal.com/uk/webapps/mpp/paypal-popup, last visit January 2023.

[102] Jack Peterson and Joseph Krug. "Augur: a decentralized, open-source platform for prediction markets". In: *CoRR* abs/1501.01042 (2015).

[103] Pierrre-Yann Dolbec. *Digital Marketing Strategy*. 2021.

[104] Plotly. *Plotly Open Source Graphing Library for Python*. [Online] https://plotly.com/python/, last visit April 2023.

[105] Polygon. *Polygon Technology*. [Online] https://polygon.technology/, last visit January 2023.

[106] Prettier. *Prettier - An Opinionated Code Formatter*. [Online] https://prettier.io/, last visit April 2023.

[107] Protofire. *Solhint*. [Online] https://github.com/protofire/solhint, last visit April 2023.

[108] Alex Pruden. *Data Availability Scaling Blockchains*. [Online] https://zeroknowledge.fm/data-availability-scaling-blockchains/, last visit December 2022.

[109] Python. *Python is a programming language that lets you work quickly and integrate systems more effectively*. [Online] https://www.python.org/, last visit April 2023.

[110] Rajagopal, Rajagopal. *Transgenerational Marketing: Evolution, Expansion, and Experience*. Jan. 2020.

[111] Daniel Ramos and Gabriel Zanko. "A Review of the Current State of Decentralized Finance as a Subsector of the Cryptocurrency Market". In: *MobileyourLife. Available at https://www. mobileyourlife. com/research* (2020).

[112] Daniel Ramos and Gabriel Zanko. "A Review of the Current State of Decentralized Finance as a Subsector of the Cryptocurrency Market". In: *MobileyourLife. Available at https://www. mobileyourlife. com/research* (2020).

[113] Siraj Raval. *Decentralized applications: harnessing Bitcoin's blockchain technology*. 2016.

[114] ReferralCandy. *Grow your sales by word-of-mouth marketing*. [Online] https://www.referralcandy.com, last visit January 2023.

[115] RefToken. *The World's First Decentralized Affiliate Platform*. [Online] https://reftoken.io/, last visit January 2023.

[116] Revolut. *Getting started with Referral Campaign*. [Online] https://help.revolut.com/help/referrals/getting-started-with-referral-campaign, last visit January 2023.

[117] Revolut. *One app, all things money.* [Online] `https://www.revolut.com`, last visit January 2023.

[118] Huaxia Rui, Yizao Liu, and Andrew B. Whinston. "Chatter matters: How twitter can open the black box of online word-of-mouth". In: *ICIS 2010 Proceedings - Thirty First International Conference on Information Systems.* ICIS 2010 Proceedings - Thirty First International Conference on Information Systems. Dec. 2010.

[119] Mary-Ann Russon. *What is Revolut?* [Online] `https://www.bbc.com/news/business-47768661`, last visit January 2023.

[120] Jan Sammut and Alex Anter. *RefToken - White Paper Business Plan.* [Online] `https://reftoken.io/uploads/reftoken-whitepaper-5.6.pdf`, last visit January 2023. 2017.

[121] Artur Sawicki. "Digital marketing". In: *World Scientific News* 48 (2016), pp. 82–88.

[122] Fabian Schär. "Decentralized finance: On blockchain-and smart contract-based financial markets". In: *FRB of St. Louis Review* (2021).

[123] Philipp Schmitt, Bernd Skiera, and Christophe Van den Bulte. "Referral Programs and Customer Value". In: *Journal of Marketing* 75.1 (2011), pp. 46–59.

[124] Cosimo Sguanci, Roberto Spatafora, and Andrea Mario Vergani. "Layer 2 Blockchain Scaling: a Survey". In: *CoRR* abs/2107.10881 (2021).

[125] Jagdip Singh et al. "Sales profession and professionals in the age of digitization and artificial intelligence technologies: concepts, priorities, and questions". In: *Journal of Personal Selling Sales Management* 39 (Jan. 2019), pp. 1–21.

[126] sol2uml. *A visualisation tool for Solidity contracts.* [Online] `https://github.com/naddison36/sol2uml`, last visit April 2023.

[127] Solidity. *Solidity Documentation.* [Online] `https://soliditylang.org/`, last visit April 2023.

[128] Solidity. *Solidity Documentation- Function Overloading.* [Online] `https://docs.soliditylang.org/en/v0.8.19/contracts.html#function-overloading`, last visit April 2023.

[129] Solidity-Coverage. *Code coverage for Solidity testing.* [Online] `https://github.com/sc-forks/solidity-coverage`, last visit April 2023.

[130] Sushiswap. *Sushiswap Protocol.* [Online] `https://www.sushi.com/`, last visit May 2023.

[131] Paul Sztorc. "Truthcoin". In: *peer-to-peer oracle system and prediction marketplace.* (2015).

[132] Tobias Boner. *Deferral - Decentralized Referral Systems.* [Online] `https://github.com/dydent/Deferral`, last visit May 2023.

[133] Tobias Boner. *Deferral Visualizations.* [Online] `https://github.com/dydent/visualizations-deferral`, last visit May 2023.

[134] TypeChain. *TypeScript bindings for Ethereum smart contracts.* [Online] `https://github.com/dethcrypto/TypeChain#readme`, last visit April 2023.

[135] TypeScript. *TypeScript Documentation.* [Online] `https://www.typescriptlang.org/`, last visit April 2023.

[136] Uniswap. *Uniswap Protocol.* [Online] `https://uniswap.org/`, last visit May 2023.

[137] JP Vergne. "Decentralized vs. Distributed Organization: Blockchain, Machine Learning and the Future of the Digital Platform". In: *Organization Theory* 1.4 (2020), p. 2631787720977052.

[138] Peeter Verlegh et al. “Receiver responses to rewarded referrals: The motive inferences framework”. In: *Journal of the Academy of Marketing Science* 41 (Nov. 2013).

[139] Wikipedia. *Blockchain Oracle*. [Online] `https://en.wikipedia.org/wiki/Blockchain_oracle`, last visit January 2023.

[140] Wikipedia. *Centralisation*. [Online] `https://en.wikipedia.org/wiki/Centralisation`, last visit January 2023.

[141] Wikipedia. *Client-Server Model*. [Online] `https://en.wikipedia.org/wiki/Client-server_model`, last visit January 2023.

[142] Wikipedia. *Decentralization*. [Online] `https://en.wikipedia.org/wiki/Decentralization`, last visit January 2023.

[143] Wikipedia. *Decentralized Systems*. [Online] `https://en.wikipedia.org/wiki/Decentralised_system`, last visit January 2023.

[144] Wikipedia. *Peer-to-peer*. [Online] `https://en.wikipedia.org/wiki/Peer-to-peer`, last visit January 2023.

[145] Wikipedia. *Referral Marketing*. [Online] `https://en.wikipedia.org/wiki/Referral_marketing`, last visit December 2022.

[146] Wikipedia. *Web3*. [Online] `https://en.wikipedia.org/wiki/Web3`, last visit January 2023.

[147] Brian C. Williams and Christopher R. Plouffe. “Assessing the evolution of sales knowledge: A 20-year content analysis”. In: *Industrial Marketing Management* 36.4 (2007), pp. 408–419.

[148] Gavin Wood. “Ethereum: A Secure Decentralised Generalised Transaction Ledger”. In: *Ethereum Project Yellow Paper* Berlin Version Beacfbd (2022), p. 41.

[149] Ping Xiao, Christopher S. Tang, and Jochen Wirtz. “Optimizing referral reward programs under impression management considerations”. In: *European Journal of Operational Research* 215.3 (2011), pp. 730–739.

[150] Junfeng Xie et al. “A Survey on the Scalability of Blockchain Systems”. In: *IEEE Network* 33.5 (2019), pp. 166–173.

[151] Yarn. *Safe, stable, reproducible projects*. [Online] `https://yarnpkg.com/`, last visit April 2023.

[152] Shijie Zhang and Jong-Hyouk Lee. “Double-Spending With a Sybil Attack in the Bitcoin Decentralized Network”. In: *IEEE Transactions on Industrial Informatics* 15.10 (2019), pp. 5715–5722.

[153] Qiheng Zhou et al. “Solutions to Scalability of Blockchain: A Survey”. In: *IEEE Access* 8 (2020), pp. 16440–16455.

Abbreviations

AMA	American Marketing Association
API	Application Programming Interface
CI	Continuous Integration
CLI	Command Line Interface
CLV	Customer Lifetime Value
CRV	Customer Referral Value
DAO	Decentralized Autonomous Organization
DApp	Decentralized Application
DeFi	Decentralized Finance
DEX	Decentralized Exchange
DLT	Distributed Ledger Technology
EWOM	Electronic Word of Mouth
EVM	Ethereum Virtual Machine
Fintech	Financial Technology
ICO	Initial Coin Offering
IPFS	InterPlanetary File System
IOT	Internet of Things
LP	Liquidity Provider
NFT	Non Fungible Token
P2P	Peer To Peer
SC	Smart Contract
SEO	Search Engine Optimization
SMM	Social Media Marketing
UCG	User Generated Content
UML	Unified Modelling Language
USD	US Dollars
UX	User Experience
WOM	Word of Mouth

List of Figures

List of Tables

Listings

Appendix A

Contents of the CD

The following deliverables are submitted for this thesis:

- **Code:** ZIP file containing the source code of the two GitHub repositories [132, 133]
  - Deferral Main Repository: Contains the source code and results data from the main repository, which can also be found on GitHub [132].
  - Deferral Visualization Repository: Contains the source code from the visualization repository, which can also be found on GitHub [133]. The code is located in the `visualizations-deferral` folder of the root `Deferral` directory.
- **Thesis:**
  - PDF version of the thesis.
  - ZIP file containing the source code of the thesis.
  - Plain text files for the English and German versions of the abstract.

Appendix B

Installation Guidelines

The complete installation guidelines and additional information for the two GitHub repositories can be found in their respective `README.md` files [132, 133]. The most important steps for the installation are also listed here. For further information, refer to the repositories on GitHub and their `README.md` files.

B.1 Deferral Repository

Follow these steps to set up and run the code locally. Ensure the latest version of Node.js and yarn (using npm is also possible) are installed. Be aware that storing files programmatically, done for the log file creation in the Deferral code, differs between Windows and, Mac & Linux due to the differing syntax used for file paths. Hence, this can lead to errors when using Windows. Therefore it is recommended to use Mac or Linux.

To install the Deferral repository, follow these steps:

1. Clone the repository:

   ```
   git clone https://github.com/dydent/Deferral.git
   ```

2. Navigate into the project directory:

   ```
   cd Deferral
   ```

3. Install the dependencies:

   ```
   yarn install
   ```

B.1.1 Environment Variables

It is necessary to create a local `.env` file containing key-value pairs of environment variables the project requires.

The `.env.example` file shows examples of all the values that must be set up and includes explanations for the different values.

B.1.2 Hardhat Setup and Configuration

The Hardhat project can be configured and adapted in the `hardhat.config.ts` file. Hardhat can be used to execute the tests and the scripts. More details on this can be found in the Hardhat documentation.

B.2 Deferral Visualizations Repository

First, follow the instructions below to set up a virtual environment and install the required dependencies.

B.2.1 Prerequisites

- Python 3.x
- pip (included with Python 3.4 and later)

Be aware that file paths differ between Windows and Mac & Linux due to their differing syntax. Hence, this can lead to errors in the code and scripts when using Windows. Therefore it is recommended to use Mac or Linux.

B.2.2 Setup Submodule Repository

1. Clone this repository:

```
git clone https://github.com/dydent/visualizations-deferral
```

Ensure this repository is included and located in the root folder of the Deferral main repository [132]. To set up the main repository, follow the instructions in the corresponding `README.md` file.

This repository can also be located in a different folder. However, the file and folder paths used in the code must then be adjusted.

B.2.3 Setup Virtual Environment

1. Open a terminal/command prompt and navigate to the project directory.
2. Run the following command to create a virtual environment:
   - For Linux and macOS:

     `python3 -m venv myenv`
   - For Windows:

     `py -m venv myenv`

   Replace `myenv` with the desired name for your virtual environment directory.
3. Activate the virtual environment:
   - For Linux and macOS:

     `source myenv/bin/activate`
   - For Windows:

     `myenv\Scripts\activate.bat`

   Make sure to replace `myenv` with the name of your virtual environment directory.

B.2.4 Installing Dependencies

With the virtual environment activated, run the following command to install the required packages from the `requirements.txt` file:

```
pip install -r requirements.txt
```

B.2.5 Environment Variables

It is necessary to create a local `.env` file containing key-value pairs of environment variables the project requires.

The `.env.example` file shows examples of all the values that must be set up and includes explanations for the different values.

Appendix C

Code and Documentation

All relevant source code files and other corresponding solution files can be found within the Deferral GitHub repository [132] and the Deferral visualizations repository [133]. Other than that, this section displays several code listing examples that were discussed in Chapter 5. The listings are sorted by their corresponding solution smart contract.

C.1 V1ReferralPaymentTransmitter

```
contract V1ReferralPaymentTransmitter is Ownable {
    [...]
    // constructor function to initialize receiver address and the referral
       ↪ amounts --> i.e. referral conditions
    constructor(
        address payable _receiver,
        uint256 _amount,
        uint256 _referralReward
    ) {
        require(
            _amount > _referralReward,
            "reward must be portion of paymentAmount"
        );
        receiver = _receiver;
        paymentAmount = _amount;
        referralReward = _referralReward;
    }
    [...]
}
```

Listing C.1: `constructor` Function of V1ReferralPaymentTransmitter

```
contract V1ReferralPaymentTransmitter is Ownable {
    [...]
    // function to update the referral payment amount
    function updatePaymentAmount(uint256 _newPaymentAmount) public
       ↪ onlyOwner {
        require(
```

```
6               referralReward < _newPaymentAmount,
7               "reward must be portion of paymentAmount"
8           );
9           uint256 oldPaymentAmount = paymentAmount;
10          paymentAmount = _newPaymentAmount;
11          emit PaymentAmountUpdated(oldPaymentAmount, _newPaymentAmount);
12      }
13      [...]
14  }
```

Listing C.2: `updatePaymentAmount` Function of V1ReferralPaymentTransmitter

C.2 V1ReferralPaymentQuantityUpgradable

```
1   contract V1ReferralPaymentQuantityUpgradable is Initializable,
        ↪ OwnableUpgradeable {
2       [...]
3       function updateReferralReward(uint256 _newReferralReward) public
            ↪ onlyOwner {
4           require(
5               _newReferralReward >= 0 && _newReferralReward <= 100,
6               "percentage value must be between 0 and 100"
7           );
8           rewardPercentage = _newReferralReward;
9           emit RewardUpdated(_newReferralReward);
10      }
11      [...]
12  }
```

Listing C.3: `updateReferralReward` Function of V1ReferralPaymentQuantityUpgradable

```
1   contract V1ReferralPaymentQuantityUpgradable is Initializable,
        ↪ OwnableUpgradeable {
2       [...]
3       function initialize(
4           address payable _receiver,
5           uint256 _rewardPercentage,
6           uint256 _paymentsQuantityThreshold
7       ) public initializer {
8           require(
9               _rewardPercentage >= 0 && _rewardPercentage <= 100,
10              "percentage value must be between 0 and 100"
11          );
12          __Ownable_init();
13          receiverAddress = _receiver;
14          rewardPercentage = _rewardPercentage;
15          paymentsQuantityThreshold = _paymentsQuantityThreshold;
16      }
17      [...]
18  }
```

Listing C.4: `initialize` Function of V1ReferralPaymentQuantityUpgradable

C.3 V2ReferralPaymentQuantityUpgradable

```
contract V2ReferralPaymentQuantityUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    function registerReferralPayment(address payable _referrerAddress)
        ↪ external payable {
        require(msg.sender != _referrerAddress, "Sender cannot be referrer"
            ↪ );
        // get current referee process data
        ReferralProcess storage currentProcess = refereeProcessMapping[
            msg.sender
        ];
        // referral process must not be completed
        require(
            !currentProcess.referralProcessCompleted,
            "Referral process has been completed for this address"
        );
        //  set referrer address first time
        if (!currentProcess.referrerAddressHasBeenSet) {
            // update values
            currentProcess.referrerAddress = _referrerAddress;
            currentProcess.referrerAddressHasBeenSet = true;
        }
        // set and update values
        currentProcess.paymentsValue += msg.value;
        currentProcess.paymentsQuantity += 1;
        emit ReferralConditionsUpdated(msg.sender);
        // evaluate referral process and progress
        if (currentProcess.paymentsQuantity > paymentsQuantityThreshold) {
            uint256 calculatedReward = (currentProcess.paymentsValue / 100)
                ↪ *                rewardPercentage;
            if (currentProcess.referrerAddressHasBeenSet) {
                require(
                    address(this).balance >= calculatedReward,
                    "Contract has not enough funds to pay rewards"
                );
                currentProcess.referrerAddress.transfer(calculatedReward);
                currentProcess.referralProcessCompleted = true;
                emit ReferralCompleted(msg.sender, _referrerAddress);
            }
        }
        // calculate reward and payment prices
        uint256 reward = (msg.value / 100) * rewardPercentage;
        uint256 receiverAmount = msg.value - reward;
        // forward payment to receiver
        receiverAddress.transfer(receiverAmount);
    }
    [...]
}
```

Listing C.5: `registerReferralPayment` Function of V2ReferralPaymentQuantityUpgradable

C.4 V1ReferralPaymentValueUpgradable

```
contract V1ReferralPaymentValueUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    // register and forward referral payment tx
     function registerReferralPayment(
         address payable _referrerAddress
     ) external payable {
         require(msg.sender != _referrerAddress, "Sender cannot be referrer"
             ↪ );
         // get current referee process data
         ReferralProcess storage currentProcess = refereeProcessMapping[
             msg.sender
         ];
         // referral process must not be completed
         require(
             !currentProcess.referralProcessCompleted,
             "Referral process has been completed for this address"
         );
         //  set referrer address first time
         if (!currentProcess.referrerAddressHasBeenSet) {
             // update values
             currentProcess.referrerAddress = _referrerAddress;
             currentProcess.referrerAddressHasBeenSet = true;
         }
         // set and update values
         currentProcess.paymentsValue += msg.value;
         currentProcess.paymentsQuantity += 1;
         emit ReferralConditionsUpdated(msg.sender);
         // evaluate referral process and progress
         if (currentProcess.paymentsValue > paymentsValueThreshold) {
             uint256 calculatedReward = (currentProcess.paymentsValue / 100)
                 ↪  *
                  rewardPercentage;
             if (currentProcess.referrerAddressHasBeenSet) {
                 require(
                     address(this).balance >= calculatedReward,
                     "Contract has not enough funds to pay rewards"
                 );
                 currentProcess.referrerAddress.transfer(calculatedReward);
                 currentProcess.referralProcessCompleted = true;
                 emit ReferralCompleted(msg.sender, _referrerAddress);
             }
         }
         // calculate reward and payment prices
         uint256 reward = (msg.value / 100) * rewardPercentage;
         uint256 receiverAmount = msg.value - reward;
         // forward payment to receiver
         receiverAddress.transfer(receiverAmount);
     }
     [...]
}
```

Listing C.6: `registerReferralPayment` Function of V1ReferralPaymentValueUpgradable

C.5 V2ReferralPaymentValueUpgradable

```
contract V2ReferralPaymentValueUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    // register and forward referral payment tx
     function registerReferralPayment(
         address payable _referrerAddress
     ) external payable {
         require(msg.sender != _referrerAddress, "Sender cannot be referrer"
             ↪ );
         // get current referee process data
         ReferralProcess storage currentProcess = refereeProcessMapping[
             msg.sender
         ];
         // referral process must not be completed
         require(
             !currentProcess.referralProcessCompleted,
             "Referral process has been completed for this address"
         );

         //  set referrer address first time
         if (!currentProcess.referrerAddressHasBeenSet) {
             // update values
             currentProcess.referrerAddress = _referrerAddress;
             currentProcess.referrerAddressHasBeenSet = true;
         }
         // set and update values
         currentProcess.paymentsValue += msg.value;
         currentProcess.paymentsQuantity += 1;
         emit ReferralConditionsUpdated(msg.sender);
         // evaluate referral process and progress
         if (currentProcess.paymentsValue > paymentsValueThreshold) {
             uint256 calculatedReward = (currentProcess.paymentsValue / 100)
                 ↪  *
                  rewardPercentage;
             if (currentProcess.referrerAddressHasBeenSet) {
                 claimableRewardMapping[_referrerAddress] +=
                     ↪ calculatedReward;
                 emit ReferralRewardsAllocated(_referrerAddress);
                 currentProcess.referralProcessCompleted = true;
                 emit ReferralCompleted(msg.sender, _referrerAddress);
             }
         }
         // calculate reward and payment prices
         uint256 reward = (msg.value / 100) * rewardPercentage;
         uint256 receiverAmount = msg.value - reward;
         // forward payment to receiver
         receiverAddress.transfer(receiverAmount);
     }
     [...]
}
```

Listing C.7: `registerReferralPayment` Function of V2ReferralPaymentValueUpgradable

C.6 V3ReferralPaymentValueUpgradable

```
contract V3ReferralPaymentValueUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    // forward paymentAmount to the receiver and send referralReward to the
        ↪ referrerAddress
    function registerReferralPayment(
        address payable _referrerAddress
    ) external payable {
        require(msg.sender != _referrerAddress, "Sender cannot be referrer"
            ↪ );
        // get current referee process data
        ReferralProcess storage currentProcess = refereeProcessMapping[
            msg.sender
        ];
        // referral process must not be completed
        require(
            !currentProcess.referralProcessCompleted,
            "Referral process has been completed for this address"
        );

        //  set referrer address first time
        if (!currentProcess.referrerAddressHasBeenSet) {
            // update values
            currentProcess.referrerAddress = _referrerAddress;
            currentProcess.referrerAddressHasBeenSet = true;
        }
        // set and update values
        currentProcess.paymentsValue += msg.value;
        currentProcess.paymentsQuantity += 1;
        emit ReferralConditionsUpdated(msg.sender);
        // evaluate referral process and progress
        if (currentProcess.paymentsValue > paymentsValueThreshold) {
            // calculate the total reward based on the referee payment
                ↪ value/volume
            uint256 calculatedTotalReward = (currentProcess.paymentsValue /
                100) * rewardPercentage;
            require(
                address(this).balance >= calculatedTotalReward,
                "Contract has not enough funds to pay rewards"
            );

            // calculate referee & referrer rewards from total reward
            uint256 refereeReward = (calculatedTotalReward / 100) *
                refereeRewardPercentage;
            uint256 referrerReward = calculatedTotalReward - refereeReward;

            // if reward allocation is two sided send rewards to referee
            if (refereeReward > 0 && refereeRewardPercentage > 0) {
                payable(msg.sender).transfer(refereeReward);
                emit RefereeRewardsDistributed(msg.sender);
            }
            // mark process as completed
            currentProcess.referralProcessCompleted = true;
            emit ReferralCompleted(msg.sender, _referrerAddress);
```

```
51              // send rewards to referrer
52              _referrerAddress.transfer(referrerReward);
53              emit ReferrerRewardsDistributed(_referrerAddress);
54          }
55          // calculate reward and payment prices
56          uint256 reward = (msg.value / 100) * rewardPercentage;
57          uint256 receiverAmount = msg.value - reward;
58          // forward payment to receiver
59          receiverAddress.transfer(receiverAmount);
60      }
61      [...]
62  }
```

Listing C.8: `registerReferralPayment` Function of V3ReferralPaymentValueUpgradable

C.7 V1ReferralMultilevelRewardsUpgradable

```
1   contract V1ReferralMultilevelRewardsUpgradable is Initializable,
        ↪ OwnableUpgradeable {
2       [...]
3       // overload function for referral payments without a referrer address
4       function registerReferralPayment() external payable {
5           ReferralProcess storage refereeProcess = refereeProcessMapping[
6               msg.sender
7           ];
8           // check if sender is validly registered as referee --> update
                ↪ referral process data and
9           if (
10              !refereeProcess.isRoot && refereeProcess.
                    ↪ referrerAddressHasBeenSet
11          ) {
12              // update referral process with payment
13              updateReferralProcess(
14                  msg.sender,
15                  refereeProcess.parentReferrerAddress,
16                  msg.value
17              );
18              // evaluate updated referral process
19              evaluateReferralProcess(msg.sender);
20              // forward value to the receiver address
21              forwardPayment(msg.value - (msg.value / 100) * rewardPercentage
                    ↪ );
22          }
23          // else sender is root or new root referrer
24          else {
25              // if sender not yet registered as root --> register
26              if (
27                  !refereeProcess.referrerAddressHasBeenSet &&
28                  !refereeProcess.isRoot
29              ) {
30                  // register address as new root address
31                  refereeProcess.isRoot = true;
32                  emit RootReferrerRegistered(msg.sender);
33              }
```

```
34              // update data for root address
35              refereeProcess.paymentsValue += msg.value;
36              refereeProcess.paymentsQuantity += 1;
37              // forward whole payment
38              forwardPayment(msg.value);
39          }
40      }
41      [...]
42  }
```

Listing C.9: `registerReferralPayment` Function Without Parameters of V1ReferralMultilevelRewardsUpgradable

```
1  contract V1ReferralMultilevelRewardsUpgradable is Initializable,
       ↪ OwnableUpgradeable {
2      [...]
3       // register & forward payment and update referral process data
4      function registerReferralPayment(
5          address payable _referrerAddress
6      ) external payable {
7          require(msg.sender != _referrerAddress, "Sender cannot be referrer"
               ↪ );
8
9          // check preconditions for _referrerAddress
10
11         // get current referrer process data
12         ReferralProcess storage referrerProcess = refereeProcessMapping[
13             _referrerAddress
14         ];
15         // referrer address must be registered address --< root referrer or
               ↪  other registered referee
16         require(
17             referrerProcess.isRoot || referrerProcess.
                   ↪ referrerAddressHasBeenSet,
18             "Referrer must be a registered address"
19         );
20
21         // check preconditions for sender address (referee)
22
23         // get current referee process data
24         ReferralProcess storage currentProcess = refereeProcessMapping[
25             msg.sender
26         ];
27         // address cannot be root & referral process must not be completed
28         require(!currentProcess.isRoot, "Root address cannot be a referee")
               ↪ ;
29         require(
30             !currentProcess.referralProcessCompleted,
31             "Referral process has been completed for this address"
32         );
33
34         // update referral process with payment
35         updateReferralProcess(msg.sender, _referrerAddress, msg.value);
36         // evaluate updated referral process
37         evaluateReferralProcess(msg.sender);
38         // forward value to the receiver address
39         forwardPayment(msg.value - (msg.value / 100) * rewardPercentage);
```

```
    }
    [...]
}
```

Listing C.10: `registerReferralPayment` Function With Parameters of V1ReferralMultilevelRewardsUpgradable

```
contract V1ReferralMultilevelRewardsUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
     function getAllParentReferrerAddresses(
        address _referee
    ) internal view returns (address payable[] memory) {
        uint256 length = 0;
        address currentRefereeAddress = _referee;
        //        loop until get to root address
        while (refereeProcessMapping[currentRefereeAddress].isRoot != true)
            ↪ {
            length++;
            currentRefereeAddress = refereeProcessMapping[
                ↪ currentRefereeAddress]
                .parentReferrerAddress;
        }

        address payable[]
            memory parentReferrerAddresses = new address payable[](length);

        currentRefereeAddress = _referee;
        for (uint256 i = 0; i < length; i++) {
            parentReferrerAddresses[i] = refereeProcessMapping[
                currentRefereeAddress
            ].parentReferrerAddress;
            currentRefereeAddress = parentReferrerAddresses[i];
        }

        return parentReferrerAddresses;
    }
    [...]
}
```

Listing C.11: `getAllParentReferrerAddresses` Function of V1ReferralMultilevelRewardsUpgradable

C.8 V1ReferralMultilevelTokenRewardsUpgradable

```
contract V1ReferralMultilevelTokenRewardsUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
       // overload function for referral payments without a referrer
          ↪ address
    function registerReferralPayment(uint256 _paymentValue) external {
        ReferralProcess storage refereeProcess = refereeProcessMapping[
        msg.sender
        ];
        // referral process must not be completed
```

```
9           require(
10              !refereeProcess.referralProcessCompleted,
11              "Referral process has been completed for this address"
12          );
13
14          // transfer tokens from payment to this contract
15          token.transferFrom(msg.sender, address(this), _paymentValue);
16
17          // check if sender is validly registered as referee --> update
                ↪ referral process data and
18          if (
19              !refereeProcess.isRoot && refereeProcess.
                    ↪ referrerAddressHasBeenSet
20          ) {
21              // update referral process with payment
22              updateReferralProcess(
23                  msg.sender,
24                  refereeProcess.parentReferrerAddress,
25                  _paymentValue
26              );
27              // evaluate updated referral process
28              evaluateReferralProcess(msg.sender);
29
30              uint256 rewardPercentageValue = (_paymentValue *
                    ↪ rewardPercentage) /
31              100;
32              uint256 paymentValueAfterReward = _paymentValue -
33              rewardPercentageValue;
34
35              // forward value to the receiver address
36              forwardPayment(paymentValueAfterReward);
37          }
38          // else sender is root or new root referrer
39          else {
40              // if sender not yet registered as root --> register
41              if (
42                  !refereeProcess.referrerAddressHasBeenSet &&
43              !refereeProcess.isRoot
44              ) {
45                  // register address as new root address
46                  refereeProcess.isRoot = true;
47                  emit RootReferrerRegistered(msg.sender);
48              }
49              // update data for root address
50              refereeProcess.paymentsValue += _paymentValue;
51              refereeProcess.paymentsQuantity += 1;
52              // forward whole payment
53              forwardPayment(_paymentValue);
54          }
55          emit PaymentRegistered(msg.sender, _paymentValue);
56      }
57      [...]
58  }
```

Listing C.12: `registerReferralPayment` Function of V1ReferralMultilevelTokenRewardsUpgradable

```
contract V1ReferralMultilevelTokenRewardsUpgradable is Initializable,
    ↪ OwnableUpgradeable {
    [...]
    function distributeRewards(address _referee) internal {
        ReferralProcess storage refereeCompletedProcess =
            ↪ refereeProcessMapping[
        _referee
        ];

        // calculate the total reward based on the referee payment value/
            ↪ volume
        uint256 calculatedTotalReward = (refereeCompletedProcess.
            ↪ paymentsValue *
        rewardPercentage) / 100;
        require(
            token.balanceOf(address(this)) >= calculatedTotalReward,
            "Contract has not enough funds to pay rewards"
        );

        // calculate and distribute referee rewards
        uint256 refereeReward = (calculatedTotalReward *
        refereeRewardPercentage) / 100;
        token.transfer(_referee, refereeReward);
        emit RefereeRewardsDistributed(_referee, refereeReward);

        // calculate remaining referrer rewards
        uint256 referrerReward = calculatedTotalReward - refereeReward;
        // get all eligible referral addresses
        address payable[]
        memory rewardAddresses = getAllParentReferrerAddresses(_referee);

        // calculate reward per referrer in reward chain
        uint256 numberOfRewardAddresses = rewardAddresses.length;
        uint256 referrerRewardProportion = referrerReward /
        numberOfRewardAddresses;

        // distribute rewards to all eligible referrers
        for (uint256 i = 0; i < numberOfRewardAddresses; i++) {
            token.transfer(rewardAddresses[i], referrerRewardProportion);
            emit ReferralRewardsDistributed(
                rewardAddresses[i],
                referrerRewardProportion
            );
        }
    }
    [...]
}
```

Listing C.13: `distributeRewards` in V1 of the Token Reward Contract

Appendix D

Evaluation - Results and Visualizations

The following sections include further visualizations and results of the Deferral solution evaluation. Thereby, the evaluation results showing the fiat prices for the different solutions can be found in a separate section. Results and figures concerning the overall evaluation and the historical gas and price data evaluation are also shown in separate sections. For a complete overview of all result visualizations, refer to the data in the `results` directory of the Deferral repository [132].

D.1 Referral Payment Transmitter Evaluation Results

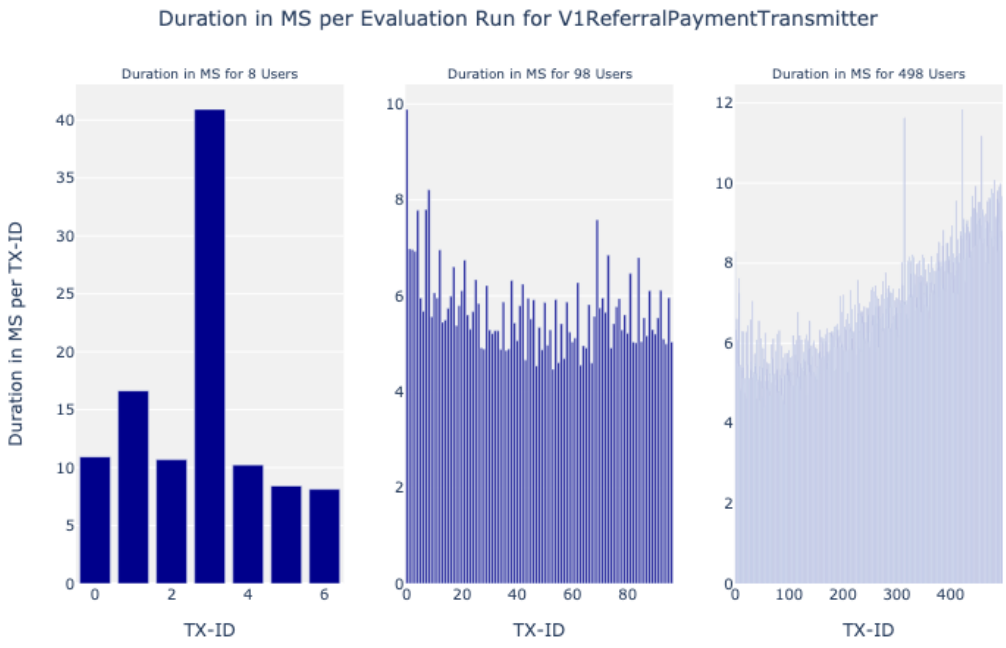


Figure D.1: DurationInMs Across the Evaluations of V1ReferralPaymentTransmitter

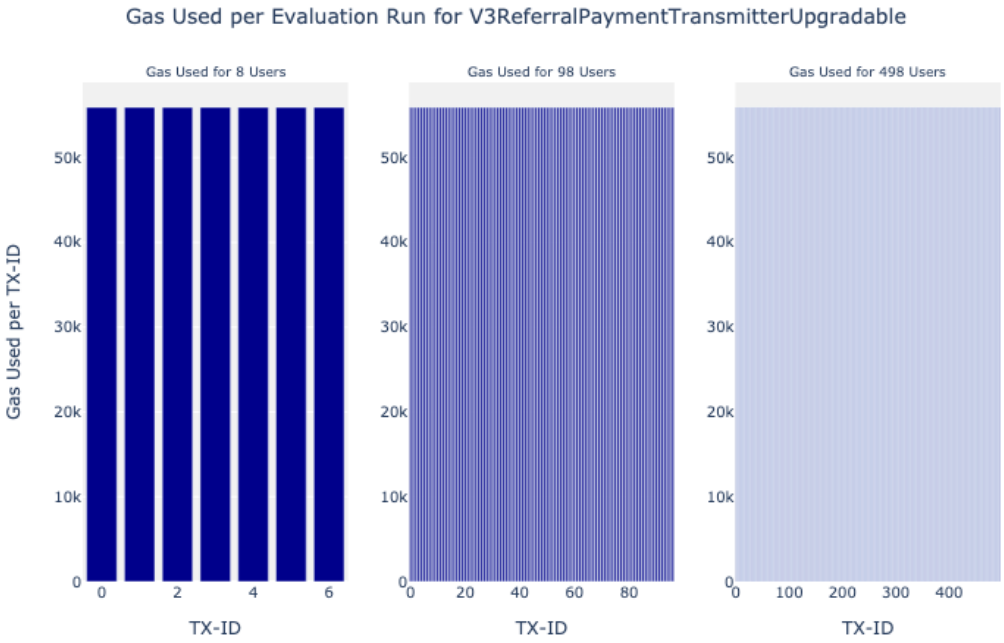


Figure D.2: GasUsed Across Evaluations of V3ReferralPaymentTransmitterUpgradable

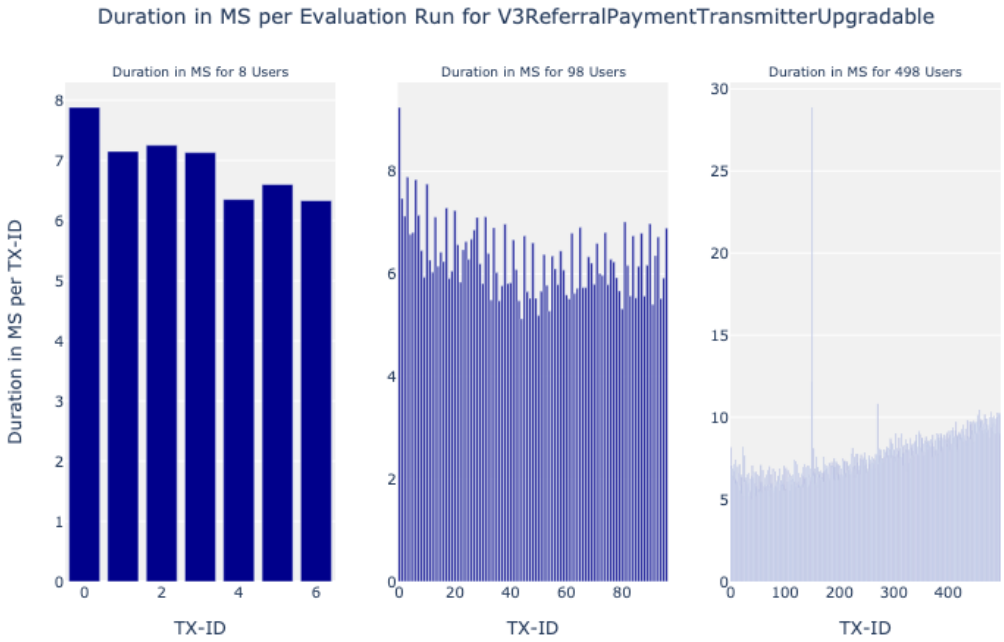
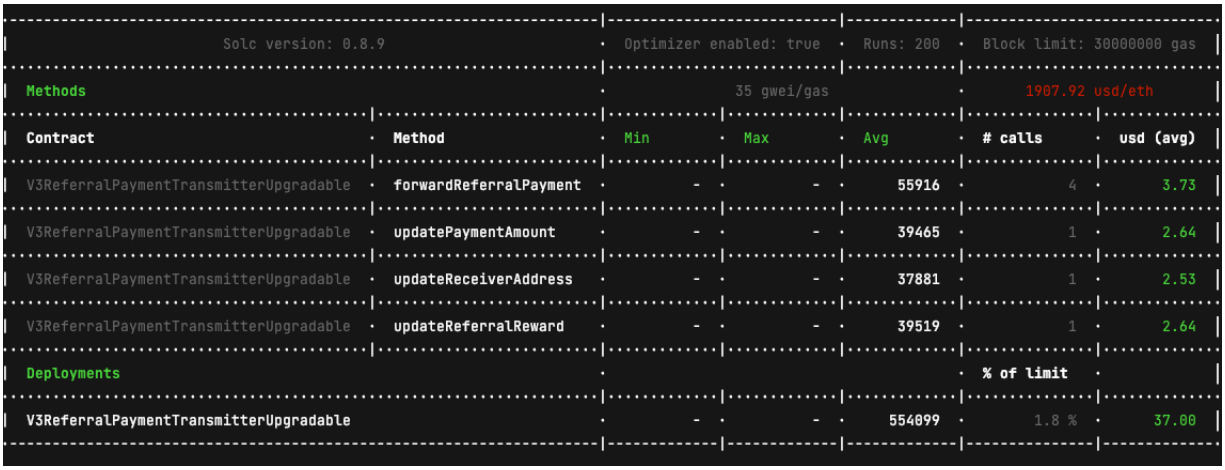


Figure D.3: DurationInMs Across Evaluations of V3ReferralPaymentTransmitterUpgradable



Solc version: 0.8.9		Optimizer enabled: true		Runs: 200	Block limit: 30000000 gas
Methods		35 gwei/gas		1907.92 usd/eth	
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
V3ReferralPaymentTransmitterUpgradable	**forwardReferralPayment**	-	-	55916	4	3.73
V3ReferralPaymentTransmitterUpgradable	**updatePaymentAmount**	-	-	39465	1	2.64
V3ReferralPaymentTransmitterUpgradable	**updateReceiverAddress**	-	-	37881	1	2.53
V3ReferralPaymentTransmitterUpgradable	**updateReferralReward**	-	-	39519	1	2.64
Deployments				**% of limit**	
V3ReferralPaymentTransmitterUpgradable		-	-	554099	1.8 %	37.00

Figure D.4: Screenshot of the Gas Reporter Output for V3ReferralPaymentTransmitterUpgradable

D.2 Referral Payment Quantity Evaluator Evaluation Results

Solc version: 0.8.9			Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas
Methods		36 gwei/gas		1907.90 usd/eth	
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
V1ReferralPaymentQuantityUpgradable	registerReferralPayment	74038	113900	100613	9	6.91
V1ReferralPaymentQuantityUpgradable	updateQuantityThreshold	-	-	37019	3	2.54
V1ReferralPaymentQuantityUpgradable	updateReceiverAddress	-	-	37527	1	2.58
V1ReferralPaymentQuantityUpgradable	updateReferralReward	32265	54177	41173	3	2.83
Deployments				**% of limit**	
V1ReferralPaymentQuantityUpgradable		-	-	750116	2.5 %	51.52

Figure D.5: Screenshot of the Gas Reporter Output for V1ReferralPaymentQuantityUpgradable

Solc version: 0.8.9			Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas
Methods		36 gwei/gas		1908.22 usd/eth	
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
V2ReferralPaymentQuantityUpgradable	registerReferralPayment	73499	113242	99994	9	6.87
V2ReferralPaymentQuantityUpgradable	updateQuantityThreshold	-	-	37019	3	2.54
V2ReferralPaymentQuantityUpgradable	updateReceiverAddress	-	-	37527	1	2.58
V2ReferralPaymentQuantityUpgradable	updateReferralReward	32265	54177	41173	3	2.83
Deployments				**% of limit**	
V2ReferralPaymentQuantityUpgradable		-	-	690041	2.3 %	47.40

Figure D.6: Screenshot of the Gas Reporter Output for V2ReferralPaymentQuantityUpgradable

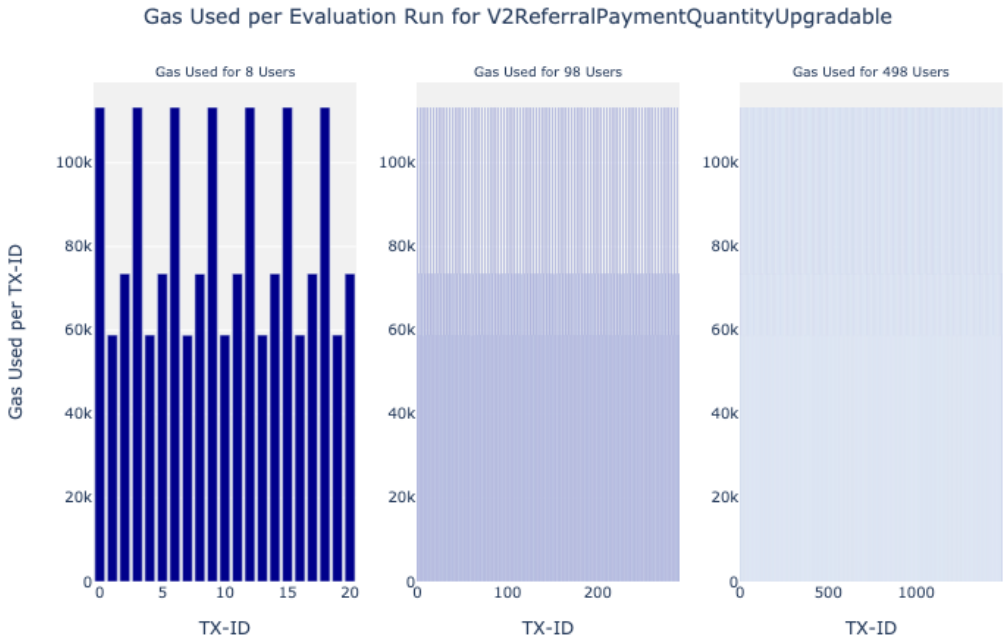


Figure D.7: GasUsed per Transaction Across the Evaluations of V2ReferralPaymentQuantityUpgradable

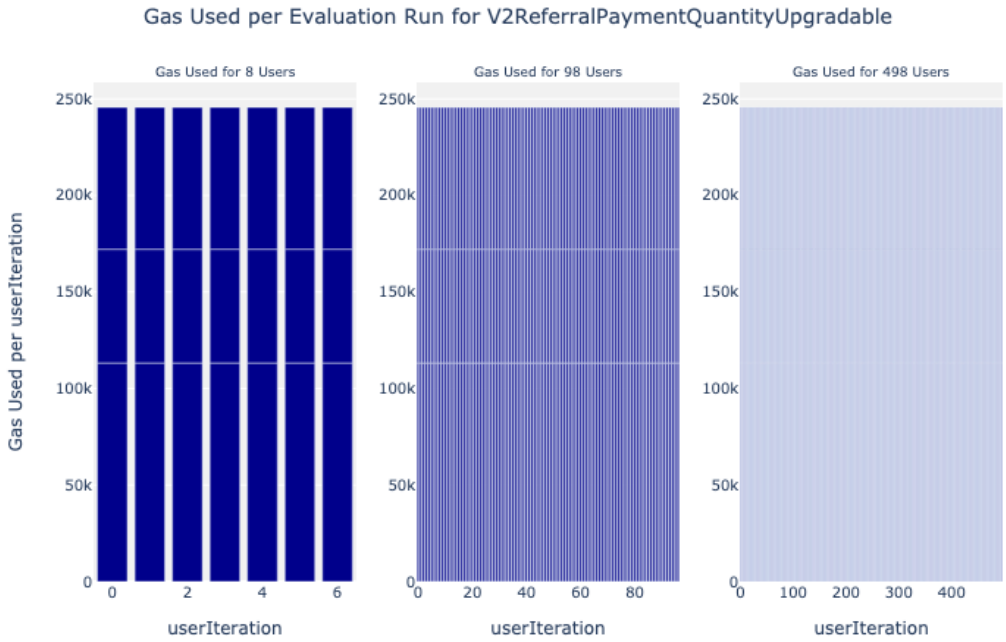


Figure D.8: GasUsed per UserIteration Across the Evaluations of V2ReferralPaymentQuantityUpgradable

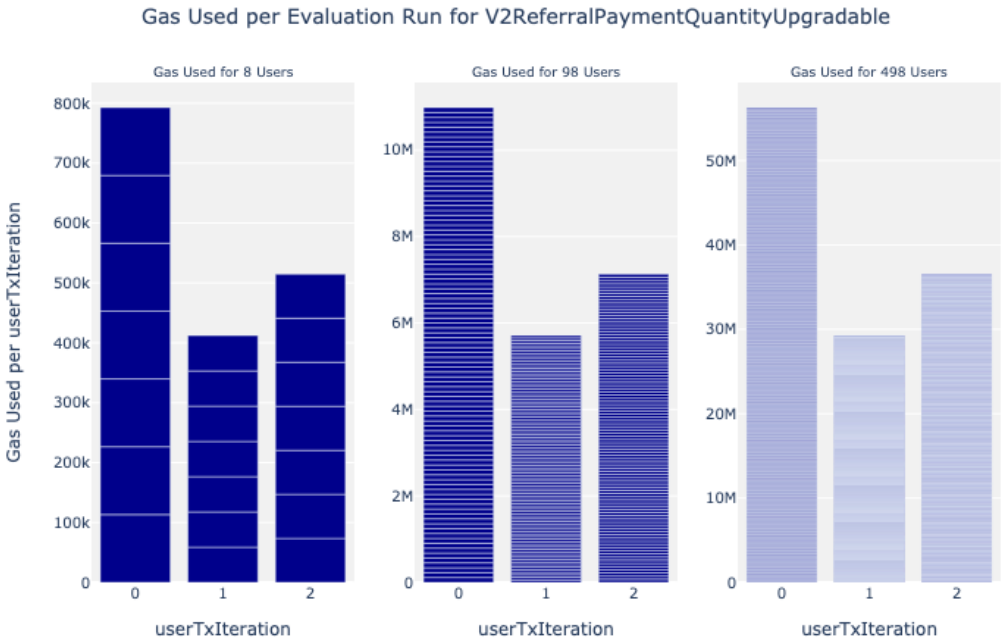
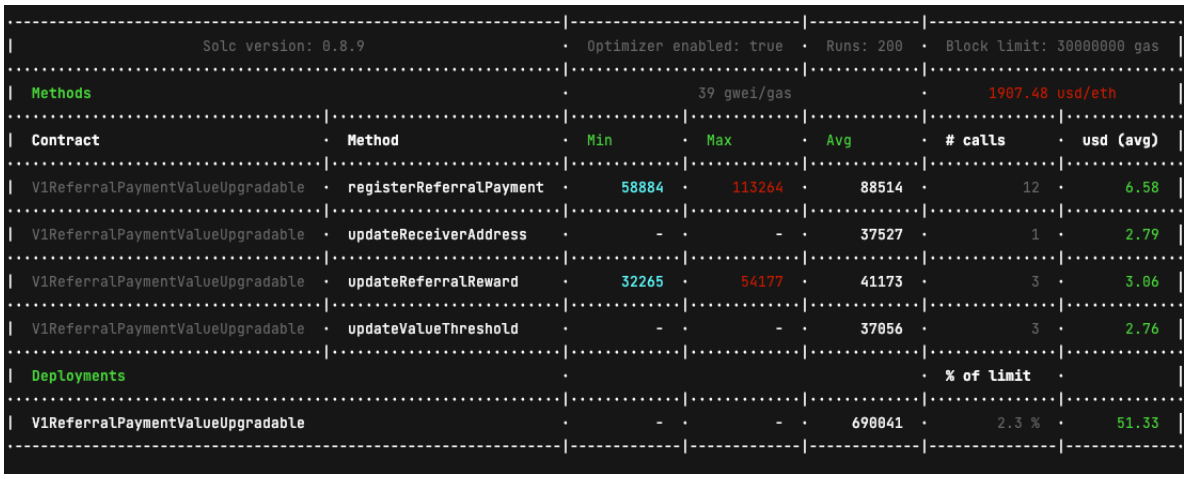


Figure D.9: GasUsed per UserTxIteration Across the Evaluations of V2ReferralPaymentQuantityUpgradable

D.3 Referral Payment Value Evaluator Evaluation Results



Solc version: 0.8.9		Optimizer enabled: true		Runs: 200	Block limit: 30000000 gas	
Methods		39 gwei/gas			1907.48 usd/eth	
Contract	Method	Min	Max	Avg	# calls	usd (avg)
V1ReferralPaymentValueUpgradable	registerReferralPayment	58884	113264	88514	12	6.58
V1ReferralPaymentValueUpgradable	updateReceiverAddress	-	-	37527	1	2.79
V1ReferralPaymentValueUpgradable	updateReferralReward	32265	54177	41173	3	3.06
V1ReferralPaymentValueUpgradable	updateValueThreshold	-	-	37056	3	2.76
Deployments					% of limit	
V1ReferralPaymentValueUpgradable		-	-	690041	2.3 %	51.33

Figure D.10: Screenshot of the Gas Reporter Output for V1ReferralPaymentValueUpgradable

Nr. of Users	Avg gasUsed	Min gasUsed	Max gasUsed	Median gasUsed
8	76 136	58 872	113 264	66 196
98	76 137	58 872	113 264	66 196
498	76 137	58 860	113 264	66 190

Table D.1: GasUsed for Evaluation Runs of V1ReferralPaymentValueUpgradable

Nr. of Users	Avg gasUsed	Min gasUsed	Max gasUsed	Median gasUsed
8	79 611	58 872	113 264	73 146
98	79 612	58 872	113 264	73 146
498	79 612	58 860	113 264	73 140

Table D.2: GasUsed for Evaluation Runs of V2ReferralPaymentValueUpgradable

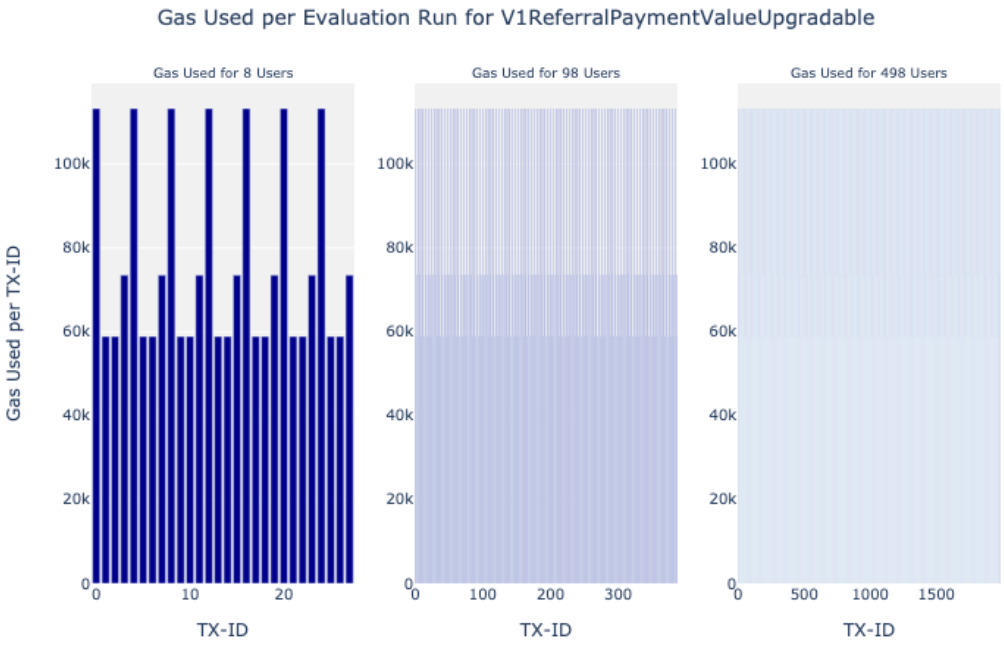


Figure D.11: GasUsed per Transaction Across the Evaluations of V1ReferralPaymentValueUpgradable

Nr. of Users	Avg gasUsed	Min gasUsed	Max gasUsed	Median gasUsed
8	78 978	58 850	113 242	71 901
98	78 978	58 850	113 242	71 901
498	78 978	58 838	113 242	71 895

Table D.3: GasUsed for Evaluation Runs of V3ReferralPaymentValueUpgradable

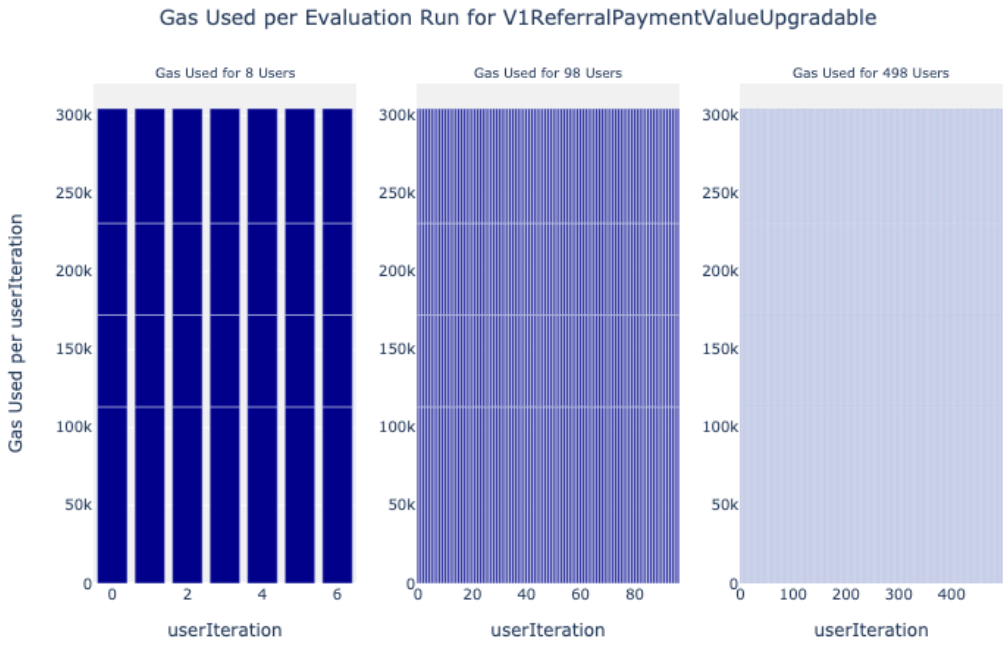


Figure D.12: GasUsed per UserIteration Across the Evaluations of V1ReferralPaymentValueUpgradable

Solc version: 0.8.9		Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas	
Methods		43 gwei/gas		1907.31 usd/eth	
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
V2ReferralPaymentValueUpgradable	claimRewards	-	-	37222	2	3.05
V2ReferralPaymentValueUpgradable	registerReferralPayment	58884	113264	90830	12	7.45
V2ReferralPaymentValueUpgradable	updateReceiverAddress	-	-	37439	1	3.07
V2ReferralPaymentValueUpgradable	updateReferralReward	32265	54177	41173	3	3.38
V2ReferralPaymentValueUpgradable	updateValueThreshold	-	-	37056	3	3.04
Deployments				**% of limit**	
V2ReferralPaymentValueUpgradable		-	-	741038	2.5 %	60.78

Figure D.13: Screenshot of the Gas Reporter Output for V2ReferralPaymentValueUpgradable

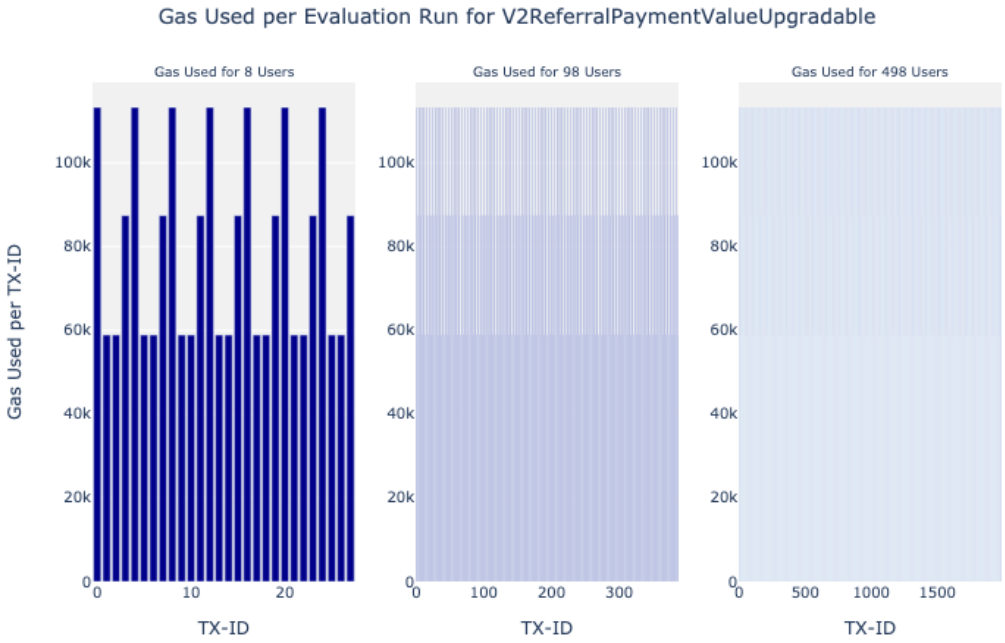


Figure D.14: GasUsed per Transaction Across the Evaluations of V2ReferralPaymentValueUpgradable

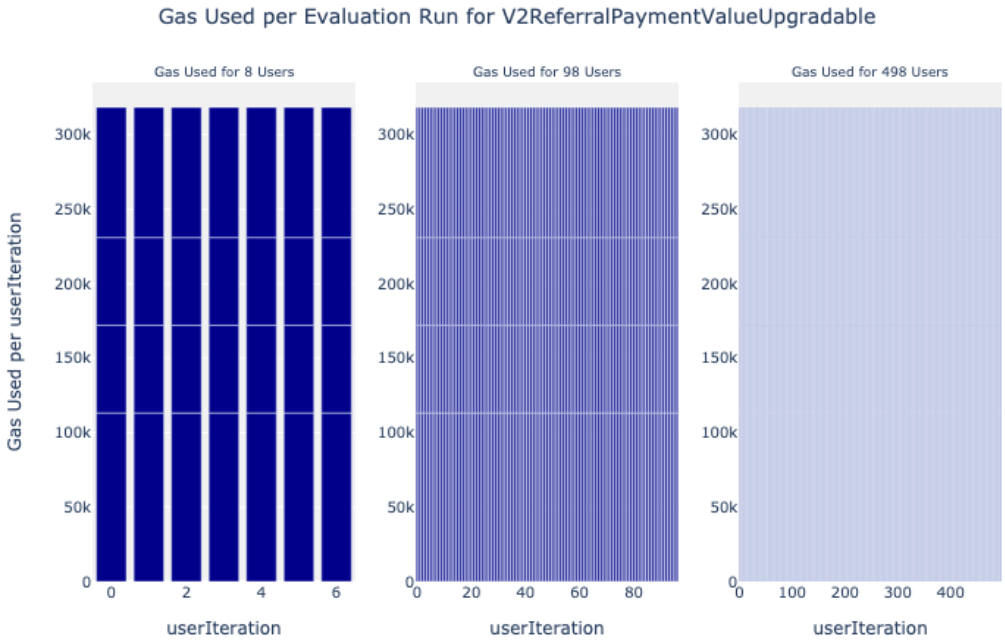


Figure D.15: GasUsed per UserIteration Across the Evaluations of V2ReferralPaymentValueUpgradable

Solc version: 0.8.9		Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas	
Methods		37 gwei/gas		1907.60 usd/eth	
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
V3ReferralPaymentValueUpgradable	registerReferralPayment	58862	113242	89981	13	6.35
V3ReferralPaymentValueUpgradable	updateReceiverAddress	-	-	37439	1	2.64
V3ReferralPaymentValueUpgradable	updateRefereeReward	32255	54167	41163	3	2.91
V3ReferralPaymentValueUpgradable	updateReferralReward	32243	54155	41151	3	2.90
V3ReferralPaymentValueUpgradable	updateValueThreshold	-	-	37134	3	2.62
Deployments				**% of limit**	
V3ReferralPaymentValueUpgradable		-	-	777322	2.6 %	54.86

Figure D.16: Screenshot of the Gas Reporter Output for V3ReferralPaymentValueUpgradable

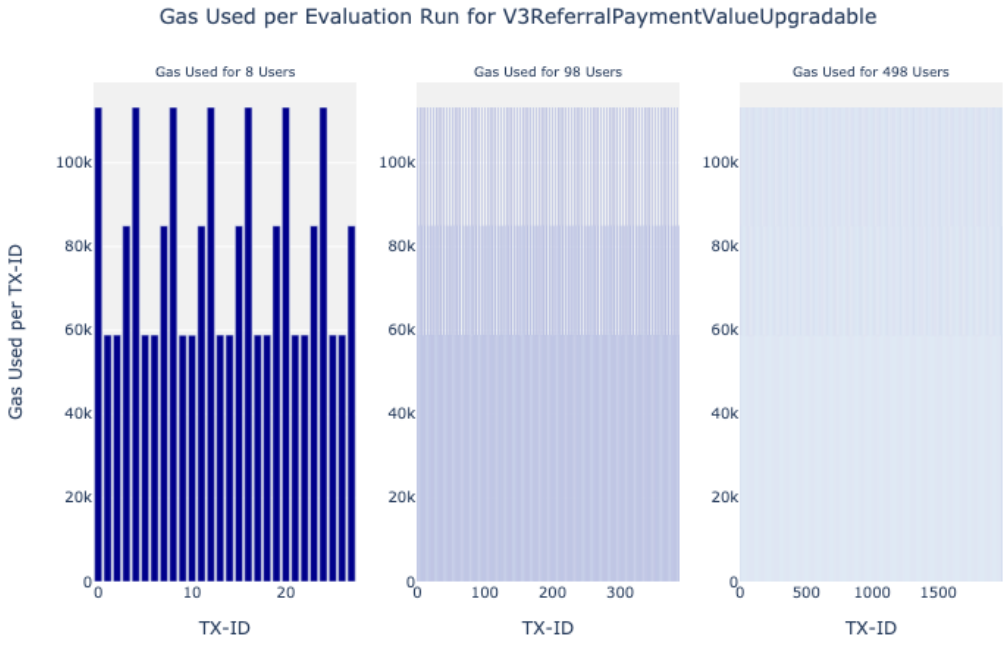


Figure D.17: GasUsed per Transaction Across the Evaluations of V3ReferralPaymentValueUpgradable

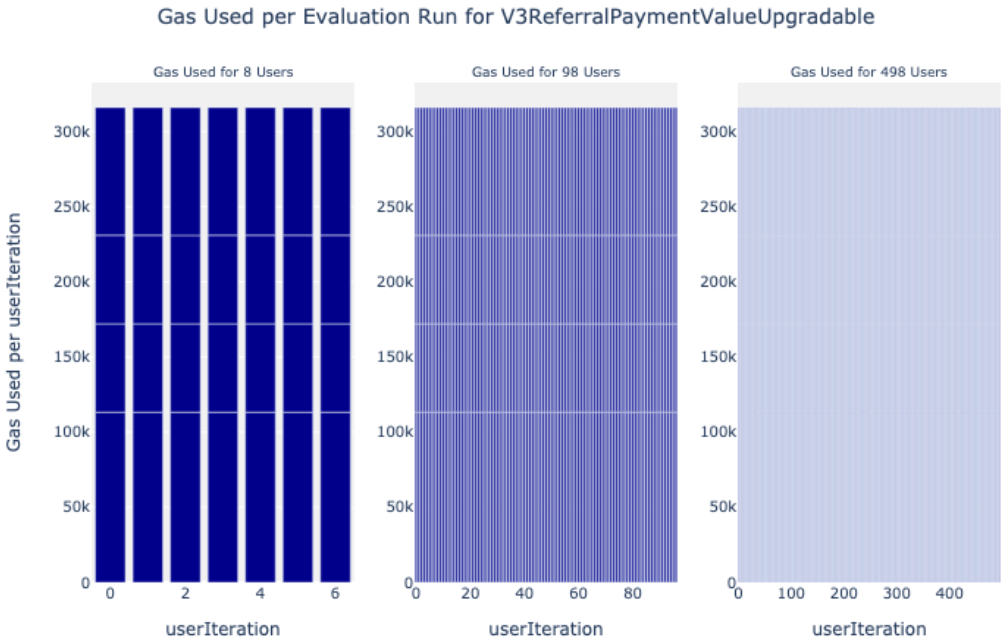


Figure D.18: GasUsed per UserIteration Across the Evaluations of V3ReferralPaymentValueUpgradable

D.4 Referral Payment Multilevel Reward Evaluator Evaluation Results

Solc version: 0.8.9		Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas	
Methods		32 gwei/gas		1909.12 usd/eth	
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
V1ReferralMultilevelRewardsUpgradable	registerReferralPayment	52792	135930	99492	31	6.08
V1ReferralMultilevelRewardsUpgradable	registerReferralPayment	64017	118507	114240	44	6.98
V1ReferralMultilevelRewardsUpgradable	updatePaymentsQuantityThreshold	-	-	37074	3	2.26
V1ReferralMultilevelRewardsUpgradable	updatePaymentsValueThreshold	37058	37070	37066	3	2.26
V1ReferralMultilevelRewardsUpgradable	updateReceiverAddress	-	-	37439	1	2.29
V1ReferralMultilevelRewardsUpgradable	updateReferralReward	32265	54177	41173	3	2.52
Deployments				**% of limit**		
V1ReferralMultilevelRewardsUpgradable		-	-	1838918	3.4 %	62.98

Figure D.19: Screenshot of the Gas Reporter Output for V1ReferralMultilevelRewardsUpgradable

Solc version: 0.8.9		Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas	
Methods		40 gwei/gas		1909.49 usd/eth	
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
V2ReferralMultilevelRewardsUpgradable	registerReferralPayment	55934	125273	104958	42	8.02
V2ReferralMultilevelRewardsUpgradable	registerReferralPayment	64594	119096	117041	54	8.94
V2ReferralMultilevelRewardsUpgradable	updateMaxRewardLevels	34278	37078	35678	2	2.73
V2ReferralMultilevelRewardsUpgradable	updatePaymentsQuantityThreshold	-	-	37096	3	2.83
V2ReferralMultilevelRewardsUpgradable	updatePaymentsValueThreshold	37058	37070	37066	3	2.83
V2ReferralMultilevelRewardsUpgradable	updateReceiverAddress	-	-	37562	1	2.87
V2ReferralMultilevelRewardsUpgradable	updateRefereeReward	32266	54178	44425	4	3.39
V2ReferralMultilevelRewardsUpgradable	updateReferralReward	32265	54177	41173	3	3.14
Deployments				**% of limit**		
V2ReferralMultilevelRewardsUpgradable		-	-	1249422	4.2 %	95.43

Figure D.20: Screenshot of the Gas Reporter Output for V2ReferralMultilevelRewardsUpgradable

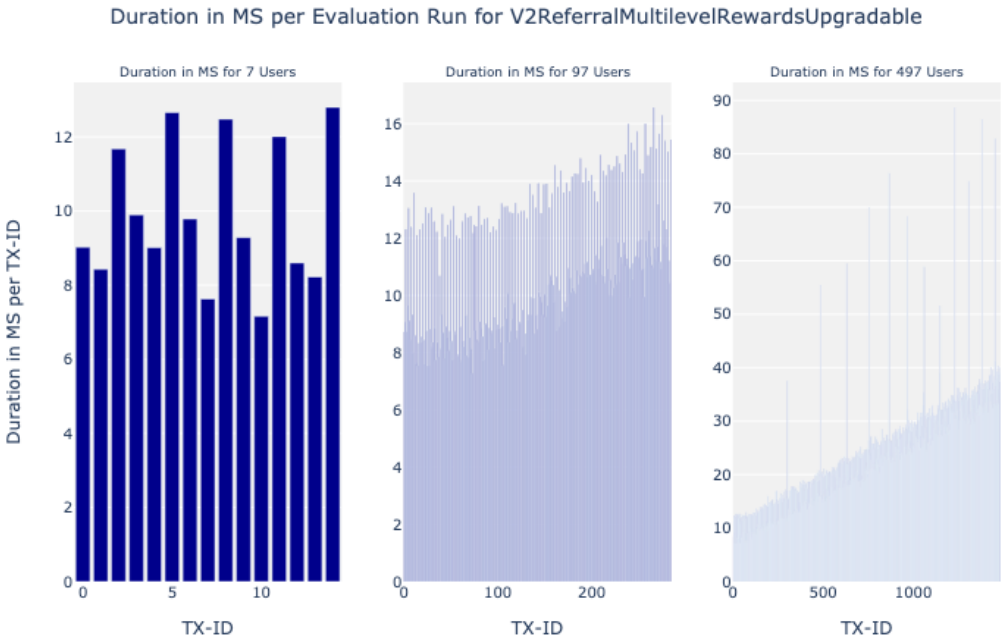


Figure D.21: DurationInMs per Transaction Across the Evaluations of V2ReferralMultilevelRewardsUpgradable

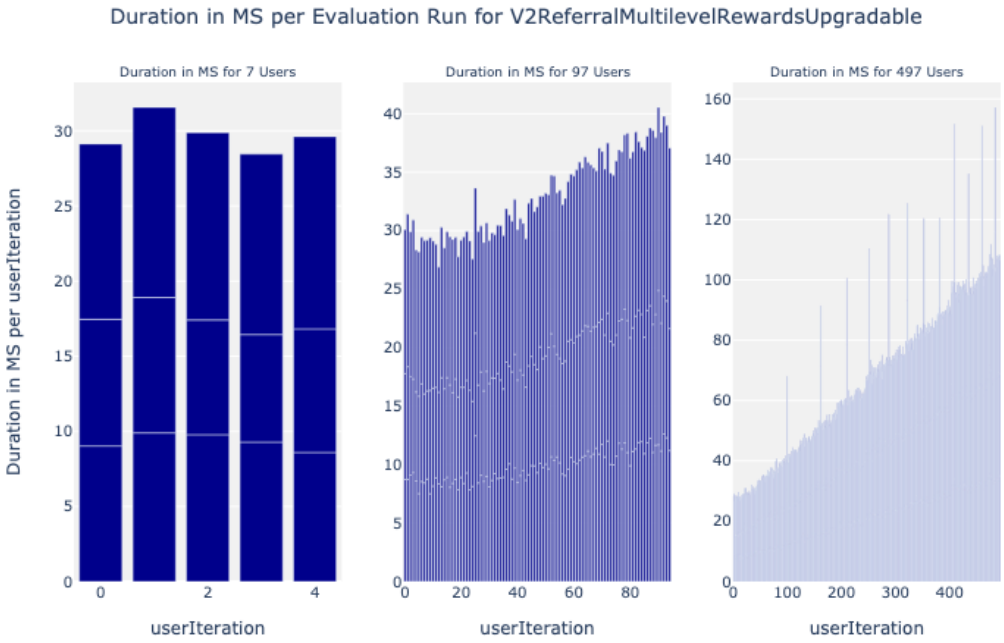


Figure D.22: DurationInMs per UserIteration for the Evaluations of V2ReferralMultilevelRewardsUpgradable

D.5 Referral Payment Multilevel Token Reward Evaluator Evaluation Results

Solc version: 0.8.9		Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas	
Methods		78 gwei/gas		1864.36 usd/eth	
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
Deferral	approve	29132	46232	44676	44	6.50
Deferral	mint	36215	53315	51604	40	7.50
V1ReferralMultilevelTokenRewardsUpgradable	registerReferralPayment	78308	145888	129877	32	18.77
V1ReferralMultilevelTokenRewardsUpgradable	registerReferralPayment	89459	160939	146387	53	21.29
V1ReferralMultilevelTokenRewardsUpgradable	updateMaxRewardLevels	34278	37078	35678	2	5.19
V1ReferralMultilevelTokenRewardsUpgradable	updatePaymentsQuantityThreshold	-	-	37096	3	5.39
V1ReferralMultilevelTokenRewardsUpgradable	updatePaymentsValueThreshold	-	-	36998	3	5.38
V1ReferralMultilevelTokenRewardsUpgradable	updateReceiverAddress	-	-	37562	1	5.46
V1ReferralMultilevelTokenRewardsUpgradable	updateRefereeReward	32266	54178	44425	4	6.46
V1ReferralMultilevelTokenRewardsUpgradable	updateReferralReward	32243	54155	41151	3	5.98
Deployments					**% of limit**	
Deferral		-	-	864122	2.9 %	125.66
V1ReferralMultilevelTokenRewardsUpgradable		-	-	1411504	4.7 %	205.26

Figure D.23: Screenshot of the Gas Reporter Output for V1ReferralMultilevelTokenRewardsUpgradable

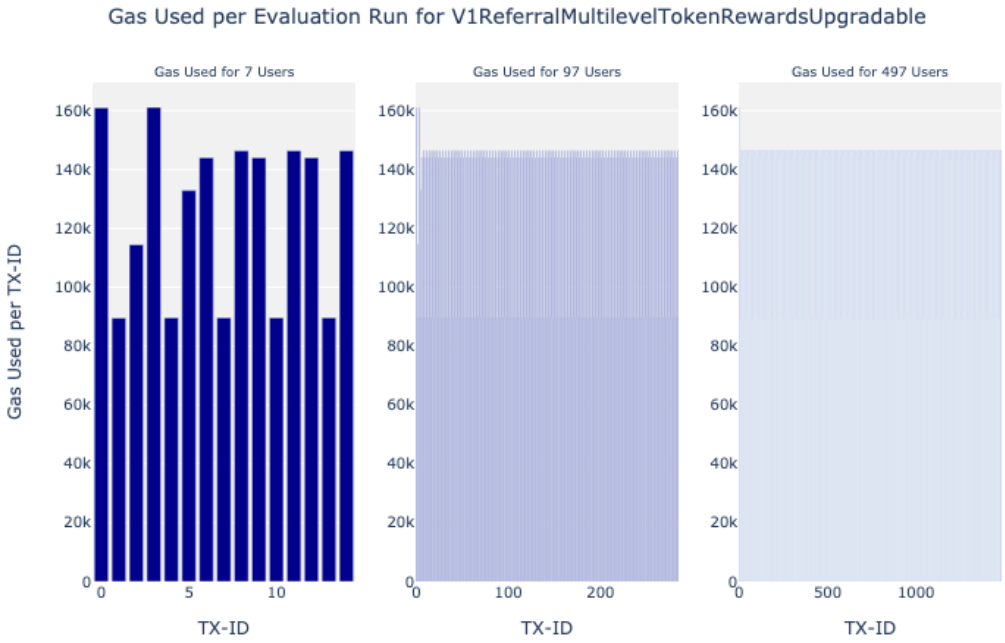


Figure D.24: GasUsed per Transaction Across the Evaluations of V1ReferralMultilevelTokenRewardsUpgradable

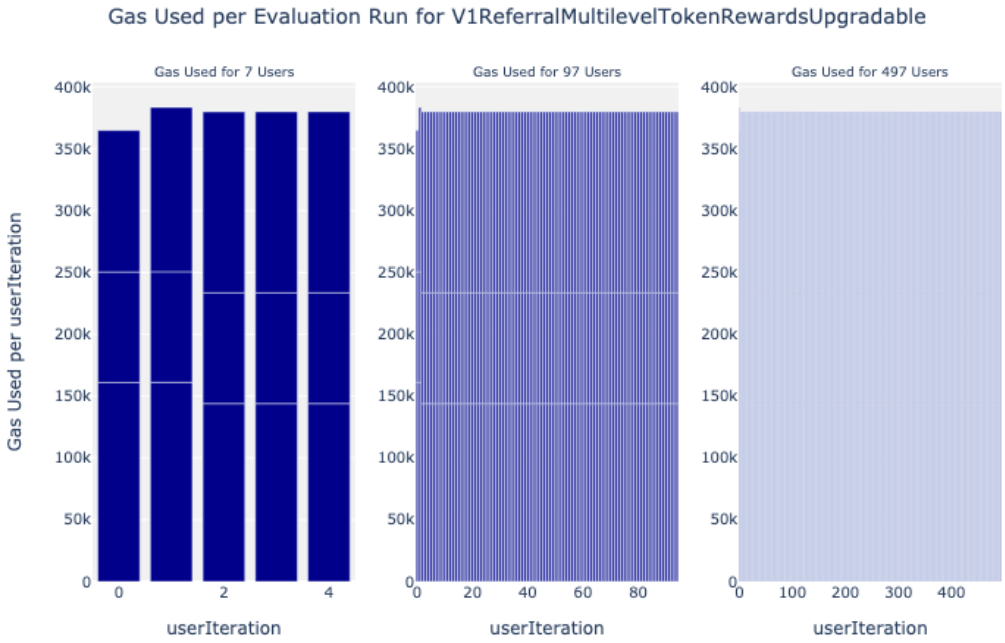


Figure D.25: GasUsed per UserIteration Across the Evaluations of V1ReferralMultilevelTokenRewardsUpgradable

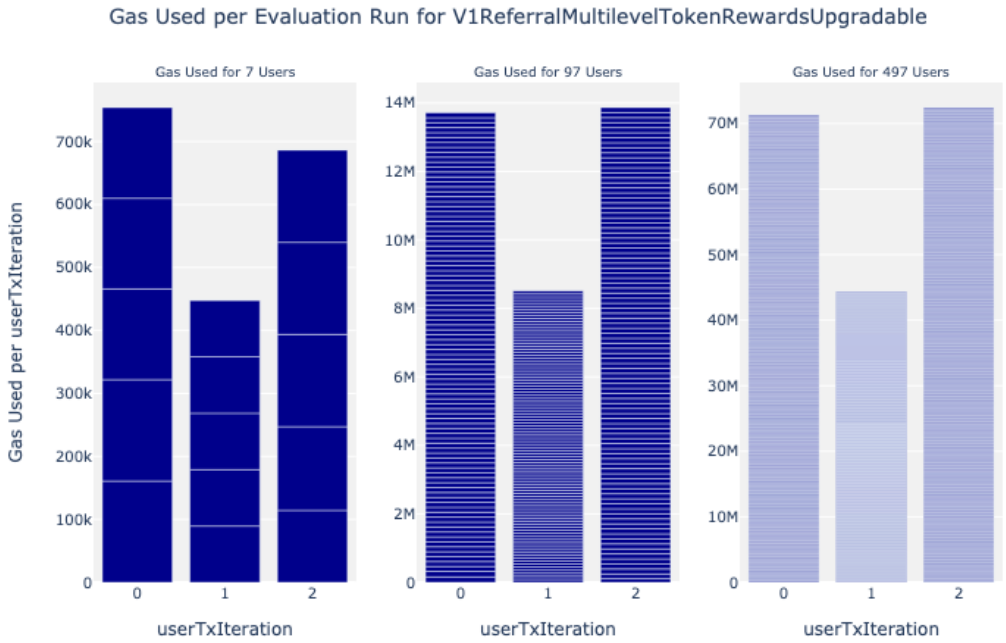


Figure D.26: GasUsed per UserTxIteration Across the Evaluations of V1ReferralMultilevelTokenRewardsUpgradable

D.6 Deferral Solutions - Fiat Costs Results

The following sections outline the results for the fiat costs of the evaluation run with 500 users for all evaluated Deferral solutions contracts. For the underlying gas prices and gas costs, as well as the complete results, refer to the complete result data in the Deferral repository [132].

Fiat Costs (in USD)	Avg	Min	Max	Median	Sum
bscFiatCost	0,047	0,047	0,047	0,047	23,528
ethereumFiatCost	7,329	7,326	7,329	7,329	3 642,544
polygonMainnetFiatCost	0,012	0,012	0,012	0,012	5,804
arbitrumMainnetFiatCost	0,009	0,009	0,009	0,009	4,587
optimismMainnetFiatCost	0,0	0,0	0,0	0,0	0,045
avalancheFiatCost	0,021	0,021	0,021	0,021	10,262
goerliFiatCost	0,006	0,006	0,006	0,006	3,117

Table D.4: Evaluation Fiat Costs Metrics (498 Users) Rounded to 3 Decimals for V1ReferralPaymentTransmitter

Fiat Costs (in USD)	Avg	Min	Max	Median	Sum
bscFiatCost	0,055	0,054	0,055	0,055	27,094
ethereumFiatCost	8,479	8,475	8,479	8,479	4 213,988
polygonMainnetFiatCost	0,014	0,014	0,014	0,014	7,07
arbitrumMainnetFiatCost	0,011	0,011	0,011	0,011	5,282
optimismMainnetFiatCost	0,0	0,0	0,0	0,0	0,052
avalancheFiatCost	0,024	0,024	0,024	0,024	11,818
goerliFiatCost	0,007	0,007	0,007	0,007	3,699

Table D.5: Evaluation Fiat Costs Metrics (498 Users) Rounded to 3 Decimals for V3ReferralPaymentTransmitterUpgradable

Fiat Costs (in USD)	Avg	Min	Max	Median	Sum
bscFiatCost	0,08	0,058	0,111	0,072	119,907
ethereumFiatCost	12,182	8,787	16,822	10,935	18 163,518
polygonMainnetFiatCost	0,021	0,015	0,029	0,019	31,151
arbitrumMainnetFiatCost	0,016	0,011	0,022	0,014	23,378
optimismMainnetFiatCost	0,0	0,0	0,0	0,0	0,232
avalancheFiatCost	0,035	0,025	0,048	0,031	52,299
goerliFiatCost	0,011	0,008	0,015	0,01	16,416

Table D.6: Evaluation Fiat Costs Metrics (498 Users) Rounded to 3 Decimals for V1ReferralPaymentQuantityUpgradable

Fiat Costs (in USD)	Avg	Min	Max	Median	Sum
bscFiatCost	0,08	0,057	0,11	0,072	119,008
ethereumFiatCost	12,447	8,946	17,218	11,175	18 558,713
polygonMainnetFiatCost	0,021	0,015	0,03	0,019	31,844
arbitrumMainnetFiatCost	0,016	0,011	0,022	0,014	23,203
optimismMainnetFiatCost	0,0	0,0	0,0	0,0	0,231
avalancheFiatCost	0,036	0,026	0,049	0,032	53,21
goerliFiatCost	0,011	0,008	0,016	0,01	17,01

Table D.7: Evaluation Fiat Costs Metrics (498 Users) Rounded to 3 Decimals for V2ReferralPaymentQuantityUpgradable

Fiat Costs (in USD)	Avg	Min	Max	Median	Sum
bscFiatCost	0,074	0,057	0,11	0,065	147,572
ethereumFiatCost	12,646	9,776	18,813	10,994	25 140,568
polygonMainnetFiatCost	0,02	0,016	0,03	0,017	39,971
arbitrumMainnetFiatCost	0,014	0,011	0,022	0,013	28,772
optimismMainnetFiatCost	0,0	0,0	0,0	0,0	0,285
avalancheFiatCost	0,032	0,025	0,048	0,028	64,517
goerliFiatCost	0,01	0,008	0,015	0,009	19,513

Table D.8: Evaluation Fiat Costs Metrics (498 Users) Rounded to 3 Decimals for V1ReferralPaymentValueUpgradable

Fiat Costs (in USD)	Avg	Min	Max	Median	Sum
bscFiatCost	0,078	0,057	0,11	0,071	154,308
ethereumFiatCost	14,523	10,737	20,661	13,342	28 871,107
polygonMainnetFiatCost	0,021	0,015	0,029	0,019	40,914
arbitrumMainnetFiatCost	0,015	0,011	0,022	0,014	30,085
optimismMainnetFiatCost	0,0	0,0	0,0	0,0	0,299
avalancheFiatCost	0,034	0,025	0,048	0,031	67,304
goerliFiatCost	0,01	0,007	0,014	0,009	18,967

Table D.9: Evaluation Fiat Costs Metrics (498 Users) Rounded to 3 Decimals for V2ReferralPaymentValueUpgradable

Fiat Costs (in USD)	Avg	Min	Max	Median	Sum
bscFiatCost	0,077	0,057	0,11	0,07	153,08
ethereumFiatCost	13,505	10,061	19,364	12,294	26 848,081
polygonMainnetFiatCost	0,02	0,015	0,029	0,018	40,099
arbitrumMainnetFiatCost	0,015	0,011	0,022	0,014	29,846
optimismMainnetFiatCost	0,0	0,0	0,0	0,0	0,296
avalancheFiatCost	0,034	0,025	0,048	0,031	66,768
goerliFiatCost	0,011	0,008	0,016	0,01	21,592

Table D.10: Evaluation Fiat Costs Metrics (498 Users) Rounded to 3 Decimals for V3ReferralPaymentValueUpgradable

Fiat Costs (in USD)	Avg	Min	Max	Median	Sum
bscFiatCost	1,196	0,06	6,749	0,113	1 776,07
ethereumFiatCost	202,967	10,225	1 145,401	19,243	301 405,738
polygonMainnetFiatCost	0,33	0,017	1,861	0,031	489,689
arbitrumMainnetFiatCost	0,233	0,012	1,316	0,022	346,294
optimismMainnetFiatCost	0,002	0,0	0,013	0,0	3,461
avalancheFiatCost	0,522	0,026	2,944	0,049	774,664
goerliFiatCost	0,153	0,008	0,865	0,015	227,659

Table D.11: Evaluation Fiat Costs Metrics (497 Users) Rounded to 3 Decimals for V1ReferralMultilevelRewardsUpgradable

Fiat Costs (in USD)	Avg	Min	Max	Median	Sum
bscFiatCost	0,101	0,063	0,123	0,116	149,511
ethereumFiatCost	16,704	10,454	20,378	19,275	24 805,322
polygonMainnetFiatCost	0,025	0,016	0,031	0,029	37,184
arbitrumMainnetFiatCost	0,02	0,012	0,024	0,023	29,135
optimismMainnetFiatCost	0,0	0,0	0,0	0,0	0,29
avalancheFiatCost	0,044	0,028	0,054	0,051	65,613
goerliFiatCost	0,013	0,008	0,016	0,015	19,591

Table D.12: Evaluation Fiat Costs Metrics (497 Users) Rounded to 3 Decimals for V2ReferralMultilevelRewardsUpgradable

Fiat Costs (in USD)	Avg	Min	Max	Median	Sum
bscFiatCost	0,124	0,087	0,157	0,14	183,465
ethereumFiatCost	20,475	14,464	26,039	23,275	30 404,734
polygonMainnetFiatCost	0,031	0,022	0,039	0,035	45,425
arbitrumMainnetFiatCost	0,026	0,018	0,033	0,03	38,841
optimismMainnetFiatCost	0,0	0,0	0,0	0,0	0,356
avalancheFiatCost	0,055	0,039	0,07	0,063	82,147
goerliFiatCost	0,014	0,01	0,018	0,016	20,512

Table D.13: Evaluation Fiat Costs Metrics (497 Users) Rounded to 3 Decimals for V1ReferralMultilevelTokenRewardsUpgradable

D.7 Overall Results and Visualizations

The following sections include a selection of the overall results of the evaluation of all the Deferral solution contracts. For further results and visualizations, refer to the GitHub repositories [132, 133].

D.7.1 Overall Gas Used Metrics

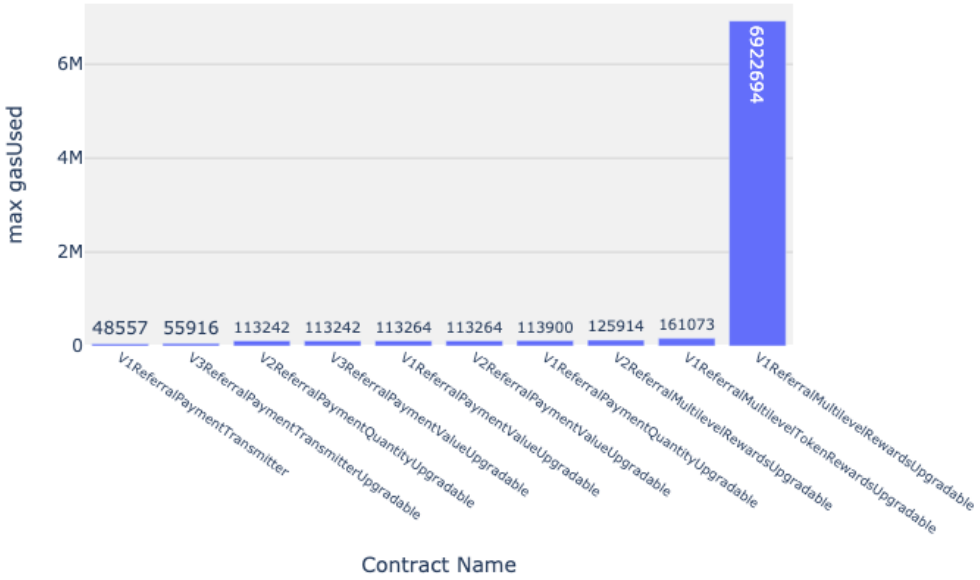


Figure D.27: Overall Max Gas Used per Transaction for All Deferral Contracts

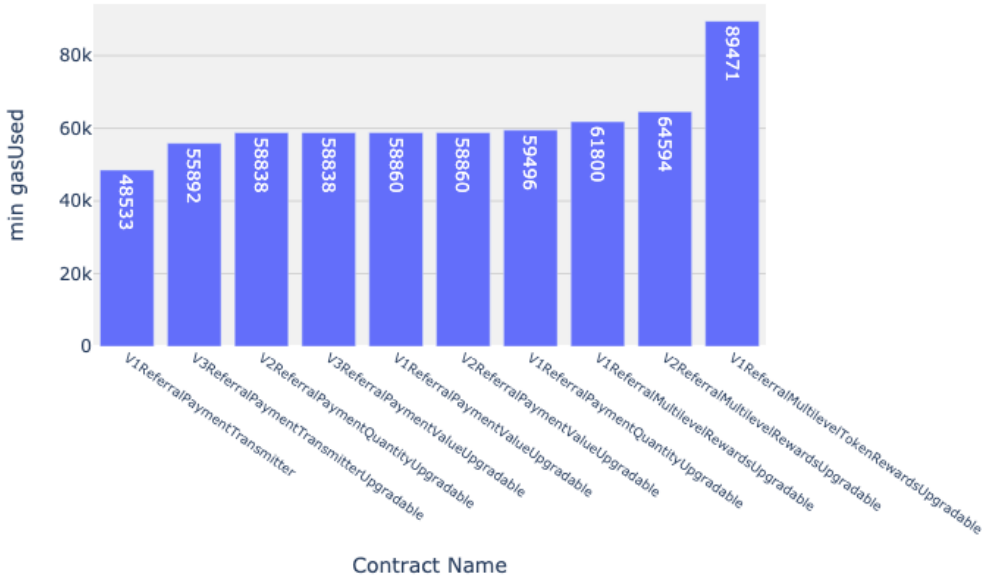


Figure D.28: Overall Min Gas Used per Transaction for All Deferral Contracts

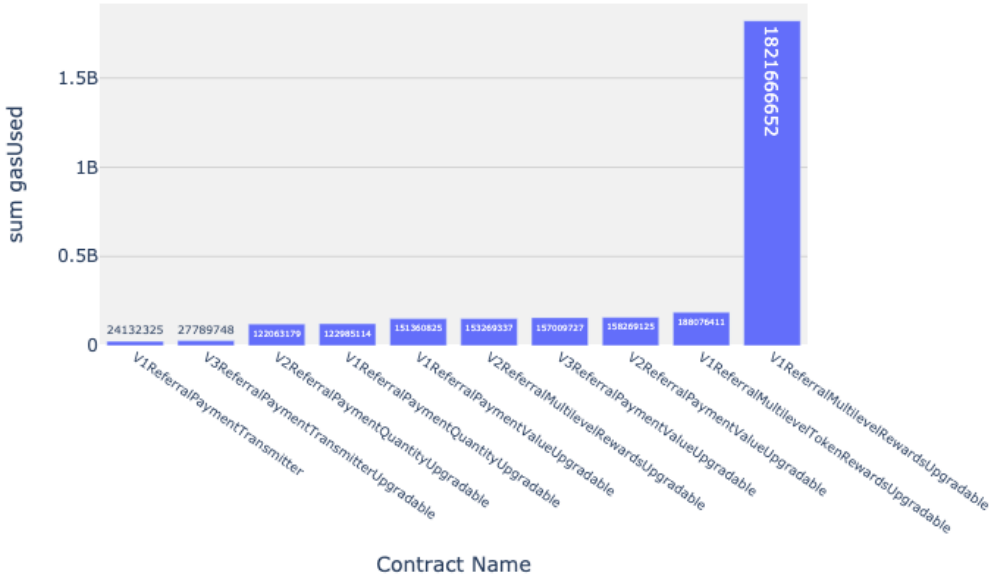


Figure D.29: Overall Sum of Gas Used per Transaction for All Deferral Contracts

D.7.2 Overall Gas Cost Metrics

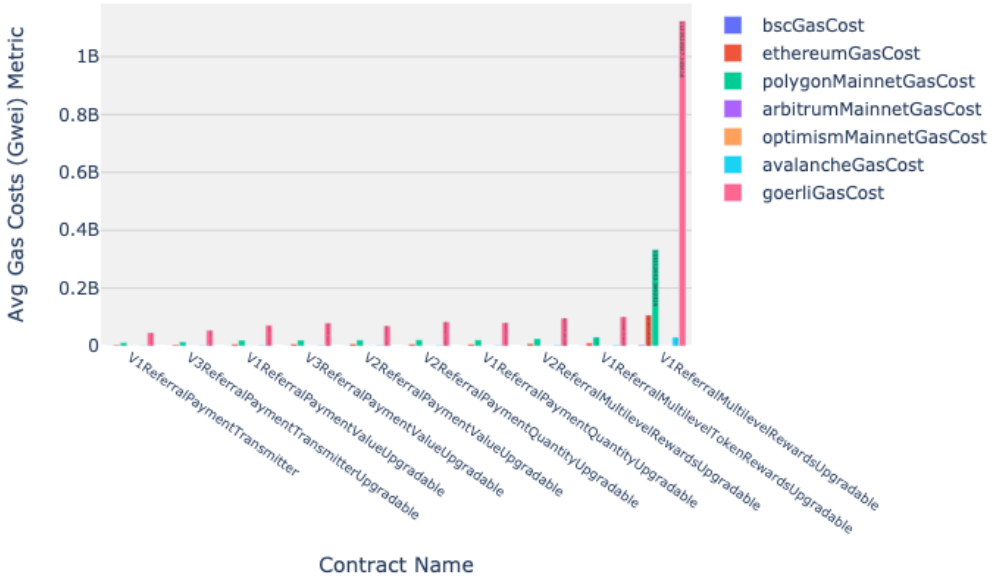


Figure D.30: Unfiltered Overall Average Gas Costs in Gwei per Contract per Evaluation Chain

Deferral Contract	Binance	Ethereum	Polygon	Arbitrum	Optimism	Avalanche	Goerli Testnet
V2ReferralMultilevelRewardsUpgradable	309 635,024	8 786 959,097	25 304 927,653	10 321,167	103,212	2 595 993,011	96 423 270,453
V1ReferralMultilevelRewardsUpgradable	3 680 134,651	106 769 575,808	333 212 853,389	122 671,155	1 226,712	30 667 788,754	1 123 520 587,346
V1ReferralMultilevelTokenRewardsUpgradable	379 952,345	10 770 477,356	30 913 533,889	13 759,341	126,651	3 250 159,906	100 954 493,724
V1ReferralPaymentTransmitter	145 667,958	3 855 413,478	11 800 159,211	4 855,599	48,556	1 213 899,648	45 960 336,86
V3ReferralPaymentTransmitterUpgradable	167 744,958	4 460 252,211	14 373 456,113	5 591,499	55,915	1 397 874,648	54 549 918,732
V2ReferralPaymentQuantityUpgradable	245 599,958	6 547 759,107	21 581 446,218	8 186,665	81,867	2 098 010,385	83 606 440,776
V1ReferralPaymentQuantityUpgradable	247 454,958	6 408 329,38	21 111 578,849	8 248,499	82,485	2 062 124,648	80 689 275,14
V1ReferralPaymentValueUpgradable	228 411,708	6 652 443,555	20 317 085,548	7 613,724	76,137	1 907 881,114	71 935 279,076
V3ReferralPaymentValueUpgradable	236 936,208	7 104 268,316	20 382 079,912	7 897,874	78,979	1 974 468,398	79 598 268,714
V2ReferralPaymentValueUpgradable	238 836,708	7 639 581,179	20 796 349,88	7 961,224	79,612	1 990 305,898	69 919 673,911

Table D.14: Avg Gas Costs in Gwei Rounded to 3 Decimals Across all Contract Evaluations with 498 Users for All Evaluation Chains

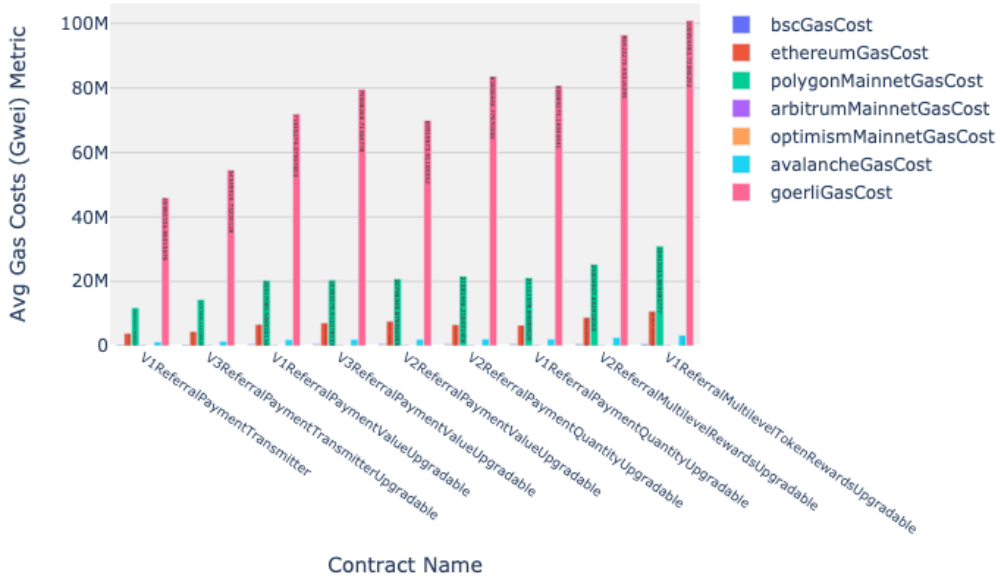


Figure D.31: Contract Filtered Overall Average Gas Costs in Gwei per Contract per Evaluation Chain

D.7.3 Overall Fiat Cost Metrics

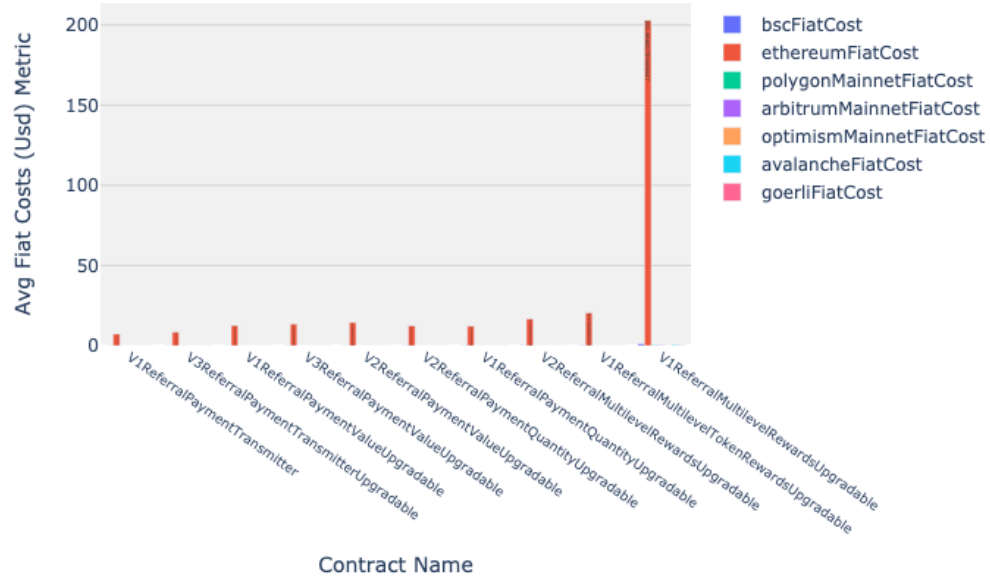


Figure D.32: Overall Average Fiat Costs in USD per Contract per Evaluation Chain

Deferral Contract	Binance	Ethereum	Polygon	Arbitrum	Optimism	Avalanche	Goerli Testnet
V2ReferralMultilevelRewardsUpgradable	0,101	16,704	0,025	0,02	0,0	0,044	0,013
V1ReferralMultilevelRewardsUpgradable	1,196	202,967	0,33	0,233	0,002	0,522	0,153
V1ReferralMultilevelTokenRewardsUpgradable	0,124	20,475	0,031	0,026	0,0	0,055	0,014
V1ReferralPaymentTransmitter	0,047	7,329	0,012	0,009	0,0	0,021	0,006
V3ReferralPaymentTransmitterUpgradable	0,055	8,479	0,014	0,011	0,0	0,024	0,007
V2ReferralPaymentQuantityUpgradable	0,08	12,447	0,021	0,016	0,0	0,036	0,011
V1ReferralPaymentQuantityUpgradable	0,08	12,182	0,021	0,016	0,0	0,035	0,011
V1ReferralPaymentValueUpgradable	0,074	12,646	0,02	0,014	0,0	0,032	0,01
V3ReferralPaymentValueUpgradable	0,077	13,505	0,02	0,015	0,0	0,034	0,011
V2ReferralPaymentValueUpgradable	0,078	14,523	0,021	0,015	0,0	0,034	0,01

Table D.15: Avg Fiat Costs in USD Rounded to 3 Decimals Across all Contract Evaluations with 498 Users for All Evaluation Chains

D.7.4 Historic Evaluation Results and Visualizations

This section includes the evaluation results and visualizations that were created by considering historical gas and fiat price data for the average recorded gas used values of all contracts for the evaluation run executed with 500 users. Not all visualizations of all Deferral solution contracts are attached here. The complete result evaluation and corresponding figures can be found in the Deferral repository [132].

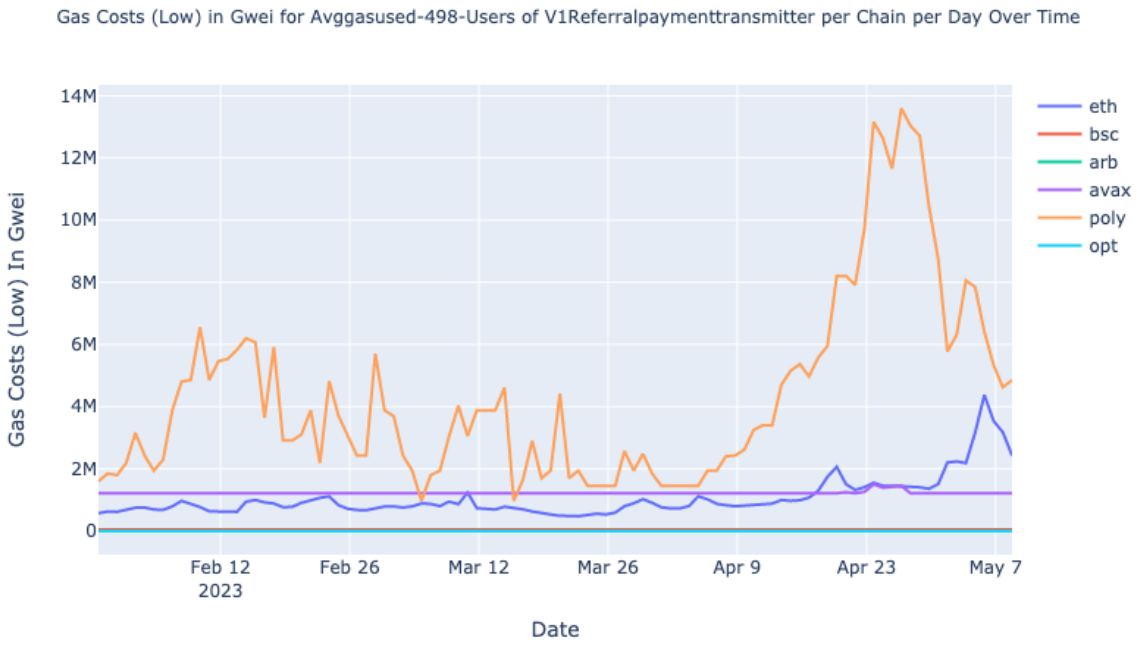


Figure D.33: Historic Average Gas Costs in Gwei Over Time for V1ReferralPaymentTransmitter

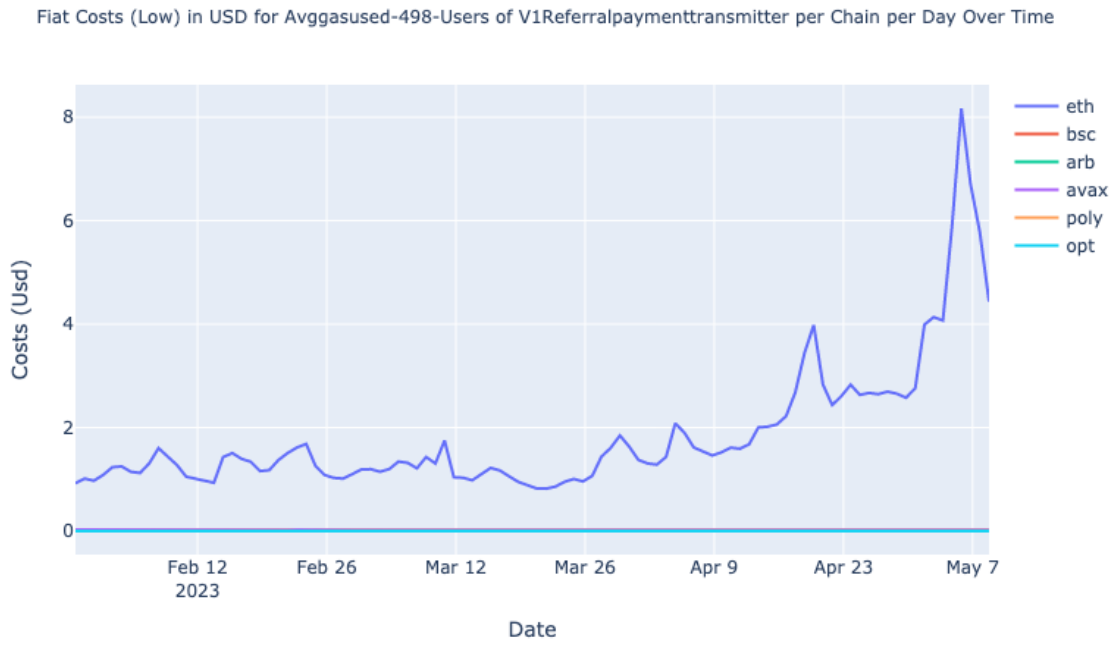


Figure D.34: Historic Average Fiat Costs in USD Over Time for V1ReferralPaymentTransmitter

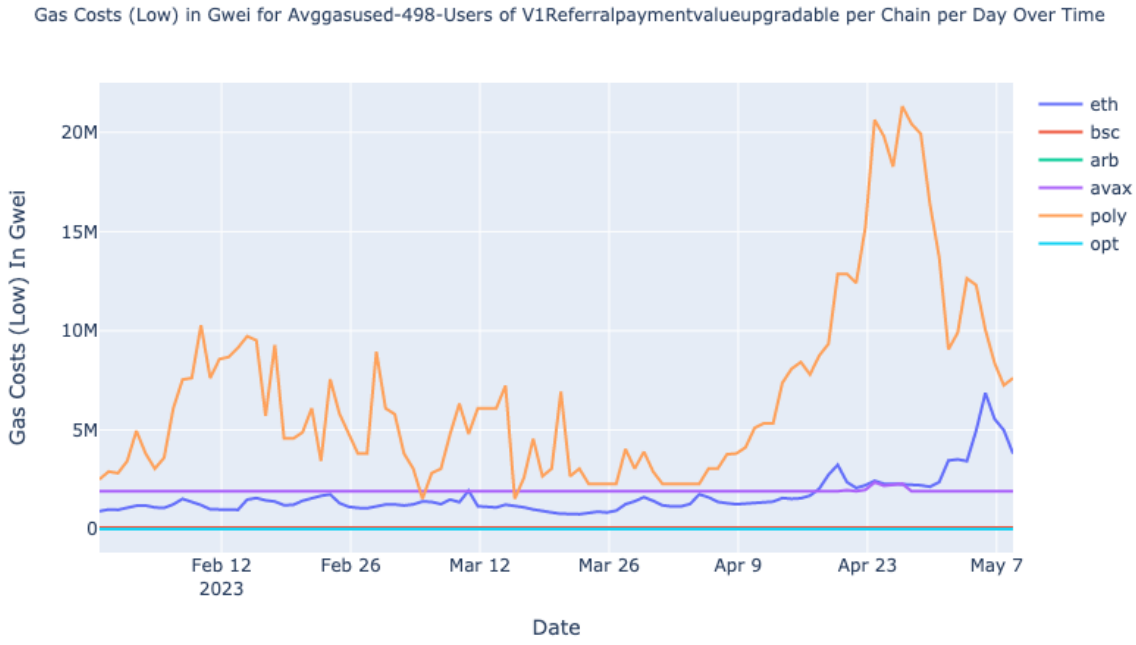


Figure D.35: Historic Average Gas Costs in Gwei Over Time for V1ReferralPaymentQuantityUpgradable

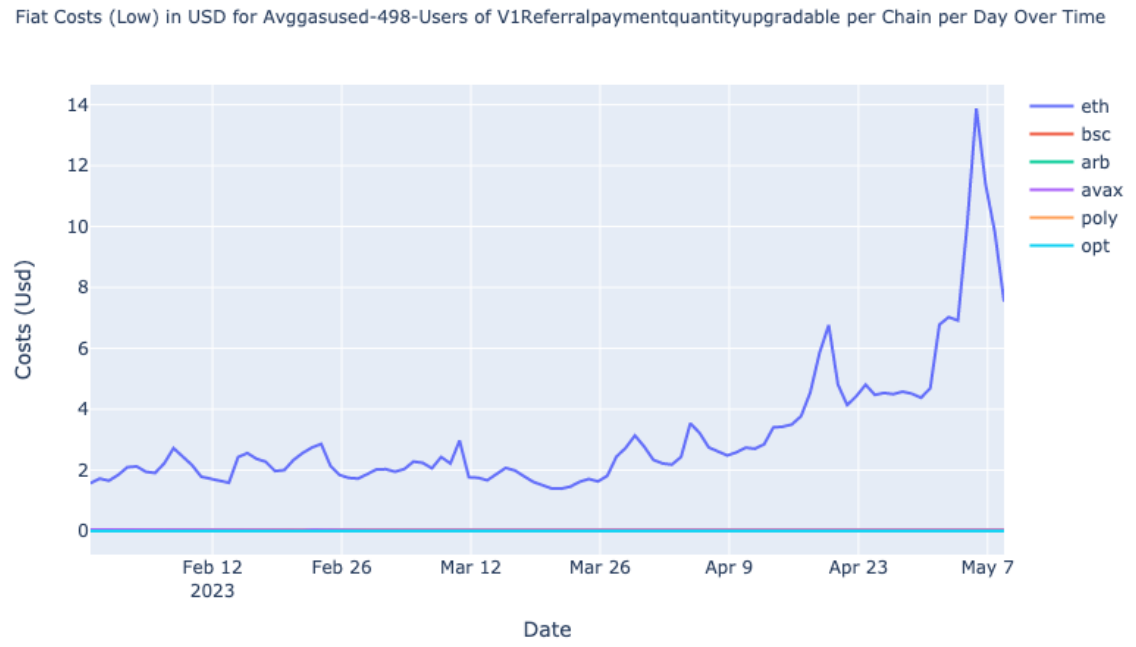


Figure D.36: Historic Average Fiat Costs in USD Over Time for V1ReferralPaymentQuantityUpgradable

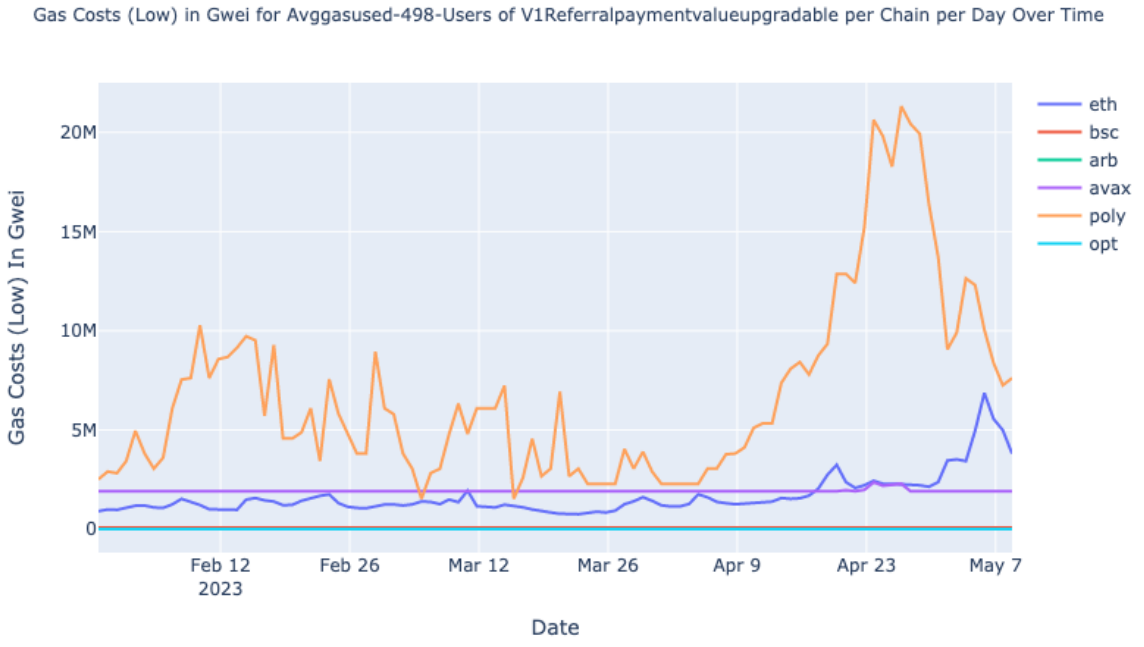


Figure D.37: Historic Average Gas Costs in Gwei Over Time for V1ReferralPaymentValueUpgradable

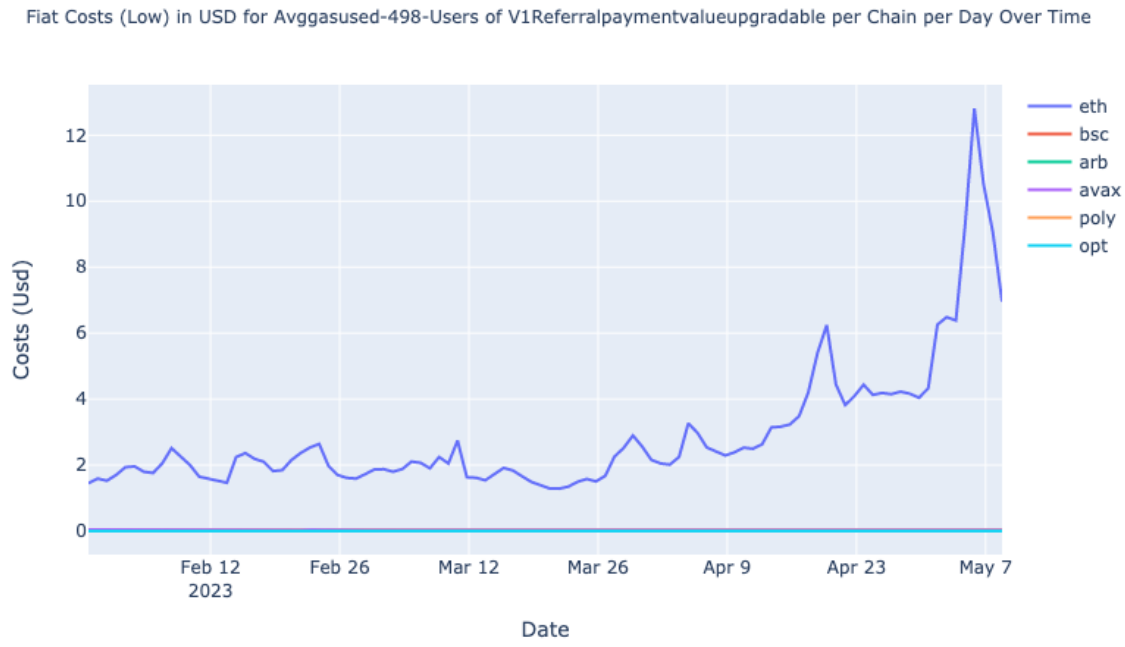


Figure D.38: Historic Average Fiat Costs in USD Over Time for V1ReferralPaymentValueUpgradable

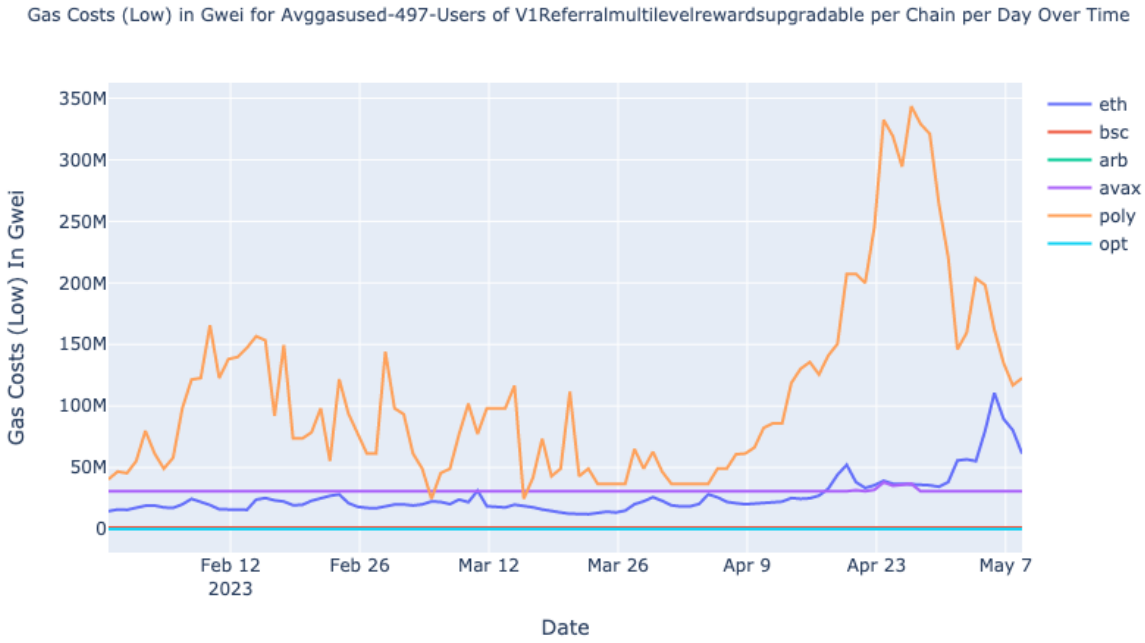


Figure D.39: Historic Average Gas Costs in Gwei Over Time for V1ReferralMultilevelRewardsUpgradable

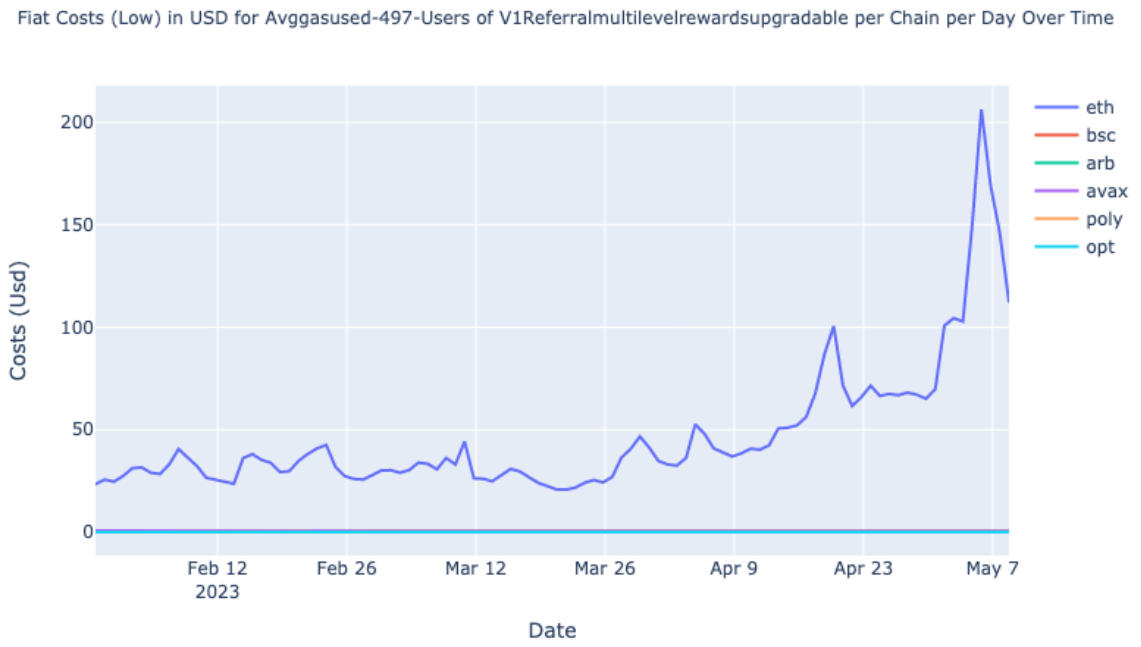


Figure D.40: Historic Average Fiat Costs in USD Over Time for V1ReferralMultilevelRewardsUpgradable

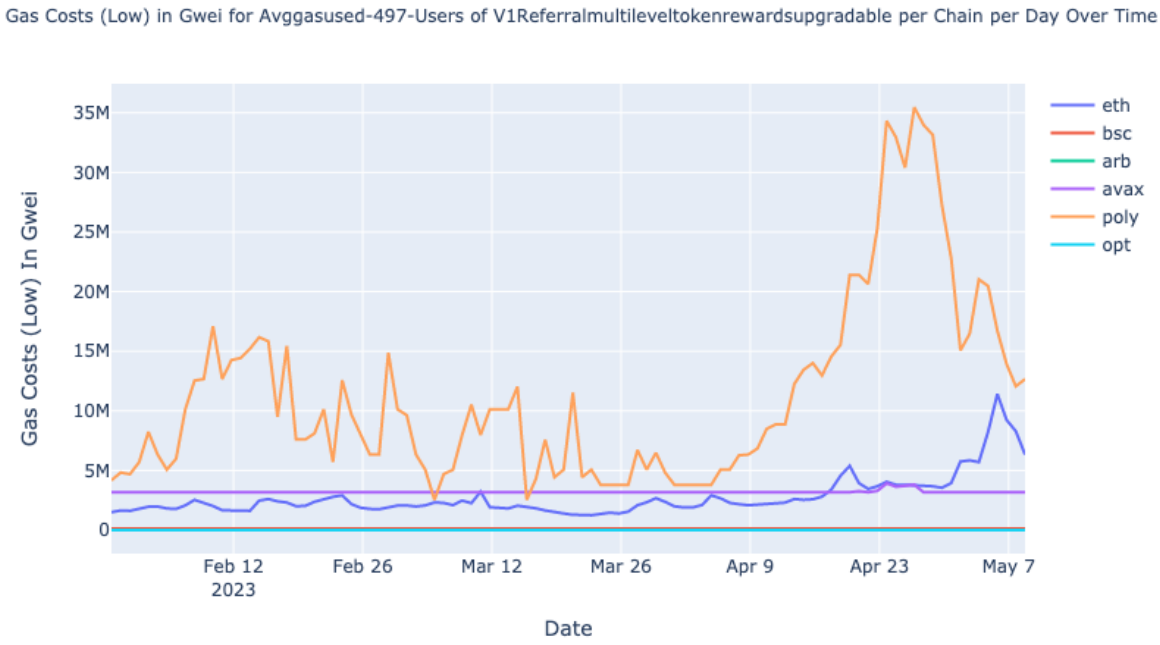


Figure D.41: Historic Average Gas Costs in Gwei Over Time for V1ReferralMultilevelTokenRewardsUpgradable

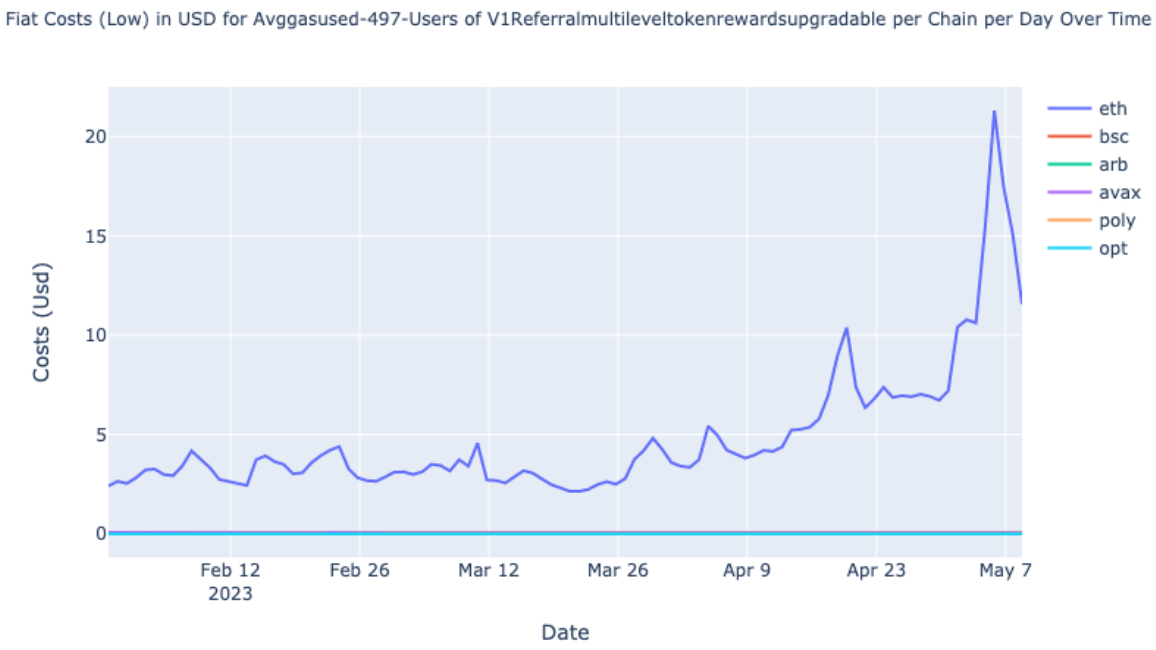


Figure D.42: Historic Average Fiat Costs in USD Over Time for V1ReferralMultilevelTokenRewardsUpgradable