EPJ Data Science
a SpringerOpen Journal

RESEARCH Open Access

DApps ecosystems: mapping the network structure of smart contract interactions

Sabrina Aufiero[1], Giacomo Ibba[2], Silvia Bartolucci[1*], Giuseppe Destefanis[3], Rumyana Neykova[3] and Marco Ortu[2]

*Correspondence:
s.bartolucci@ucl.ac.uk
[1] Dept. of Computer Science, University College London, London, UK
Full list of author information is available at the end of the article

Abstract

Decentralized applications (DApps) built on blockchain platforms such as Ethereum and coded in languages such as Solidity, have recently gained attention for their potential to disrupt traditional centralized systems. Despite their rapid adoption, limited research has been conducted to understand the underlying code structure of these applications. In particular, each DApp is composed of multiple smart contracts, each containing a number of functions that can be called to trigger a specific event, e.g., a token transfer. In this paper, we reconstruct and analyse the network of contracts and functions calls within the DApp, which is helpful to unveil vulnerabilities that can be exploited by malicious attackers. We show how decentralization is architecturally implemented, identifying common development patterns and anomalies that could influence the system's robustness and efficiency. We find a consistent network structure characterized by modular, self-sufficient contracts and a complex web of function interactions, indicating common coding practices across the blockchain community. Critically, a small number of key functions within each DApp play a central role in maintaining network connectivity, making them potential targets for cyber attacks and highlighting the need for robust security measures.

Keywords: Decentralized applications; Blockchain; Network structure; Software engineering; Smart contracts

1 Introduction

In recent years, the Total Value Locked (TVL) in decentralized finance platforms and crypto protocols has reached 44 bUSD, with over 82 million wallets and active users worldwide. Since the launch of Bitcoin, blockchains and decentralized platforms have evolved to enable new functionalities and use cases beyond digital currency. These functionalities are embedded in smart contracts, a digital agreement written in code, stored on a blockchain, and executed automatically without intermediaries [1]. Smart contracts benefit from the blockchain's security and transparency, providing users with a way to enforce agreements and streamline processes, and they are decentralized so they cannot be changed or tampered with once they are deployed. These terms can be as simple as making a single payment, or as complex as a multi-step process with many participants and data point requirements. Once deployed, anyone with access to the blockchain can invoke and interact with the smart contract. Multiple contracts can be linked together to form a

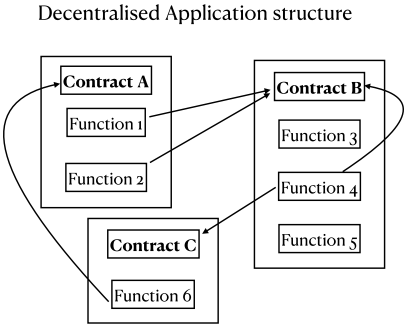


Figure 1 Scheme of a DApp structure. The DApp is composed of 3 contracts, each with a varying number of functions. Contracts can interact between them via function calls (black arrows)

more sophisticated application, with different purposes and functionalities. Notable use cases include facilitating financial transactions or gaming interactions, enabling players to own and trade in-game assets, participate in competitions, and earn rewards in the form of tokens. These decentralized applications operating on blockchain systems, also widely called DApps, enhance many traditional industries and services, and are not run or controlled by a single central authority or trusted organisation. DApps are developed in most cases in an open-source fashion on Github: they are composed of a collection of *contracts*, each containing multiple *functions* that can be called by the same or different contracts, one or multiple times, depending on the task performed by the user (see Fig. 1).

While DApps offer various advanced features like transparency and community collaboration, they may not be completely immune to security breaches or hacking attempts, and that is why a robust analytical framework is needed to study their complexity and vulnerabilities. One of the issues that users are usually concerned about is technical vulnerabilities, and while there are usually strong measures and smart contract bug bounty programs in place to address these issues [2], smart contracts can still be exploited by malicious actors, leading to financial loss or unintended consequences. As opposed to traditional finance, developers of Decentralised Finance (DeFi) projects often lack financial experience, and cyber security is an afterthought in a hasty development process [3]. Moreover, the hosting of smart contract code publicly on Github further enables an attacker's opportunity to locate vulnerabilities quickly and efficiently. Exploiting DeFi projects currently is a low-risk high-reward opportunity to malicious actors. For example in the past years, the Decentralized Finance sector has experienced thousands of attacks causing the loss of millions of dollars locked in protocols [4], which could be dramatically reduced by actively monitoring and fixing security threats related to bugs in the code. Another major challenge with DApps is scalability [5]. Some blockchains have limitations in terms of processing speed and capacity, which can result in slower transaction times and higher costs. Scalability becomes a major concern especially when the number of users and transactions increases. Often, these vulnerabilities are directly linked to the way in which contracts and functions interact.

Let us consider a toy DApp example illustrating technical vulnerabilities. In Fig. 2, we present a simplified DApp responsible for managing the buying and selling of items during an auction. There are four contracts involved: `Auction`, `Item`, `Participant`, and `Vault`. The `Auction` contract handles auction management, registering buyers, sellers, and items. The `Item` contract manages auction items and updates related to offers on an item. `Participant` manages the participants, allowing them to enrol as buyers

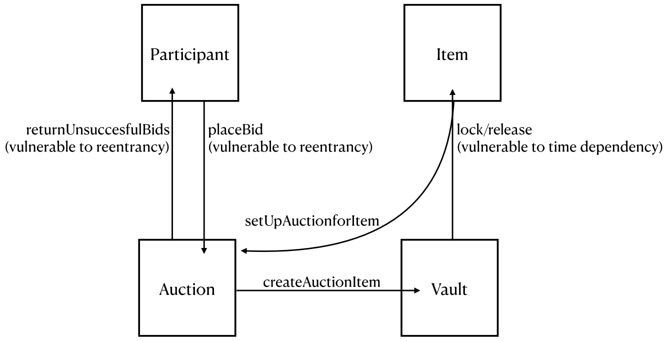


Figure 2 Scheme of a toy DApp example vulnerable to re-entrancy and time dependency attacks

or sellers and keeps track of their respective bids. Lastly, `Vault` is responsible for safeguarding items and preventing them from being assigned to anyone before the auction's end. The vulnerable functions in this context are `lock, release, placeBid`, and `returnUnsuccessfulBids` [6]. `placeBid` and `returnUnsuccessfulBids` are susceptible to reentrancy attacks. Consequently, an attacker, by reentering the function's code multiple times, could illicitly withdraw the funds related to the bids (funds meant to be collected by the auctioneer). On the other hand, the lock and release functions are vulnerable to time dependency issues, potentially causing items to be locked indefinitely or released prematurely before the end of the auction. As a result, participants may purchase items, which they will never gain ownership of, or items could be assigned to participants who did not place the highest bid during the auction. The reentrancy attack has been exploited in the famous DAO Hack, where an attacker was able to call the function SplitDAO recursively, transferring $\sim$ 50 mUSD in its account [7]. Note also that each function call has an associated computational cost to run and execute the code, and a fee paid to the network for validation (i.e., the so-called gas fees in Ethereum). In terms of scalability, a more complex call structure corresponds to higher computational costs and fees to execute a given action.

This research, leveraging complex network analysis tools, intends to study the DApps' complexity and characterize development practices. Our aim is to provide an understanding of the code structure underlying DApps deployed on different blockchains. This offers insights on architectural choices, vulnerabilities detection, and future development directions of decentralized systems.

While previous research has made significant progress in identifying vulnerabilities within individual smart contracts [6, 8], there is a gap in understanding how the complex interactions between functions across contracts can impact the overall security of DApps. Smart contracts do not operate in isolation within DApps, instead they are part of a complex interconnected ecosystem of contract, interacting via function calls, and depending on each other to perform specific tasks. These non-trivial interactions can potentially propagate vulnerabilities, and amplify their impact on the entire DApp. Traditional code analysis techniques often focus on isolated contracts and may overlook the emergent vulnerabilities arising from the architectural design and functions' interdependencies within DApps. To address this limitation, we propose a novel network-based approach to analyze the function interaction graph of DApps. By representing functions as nodes and their invocations as edges, we construct weighted directed networks that capture the flow of control and data within the DApp. This network representation allows us to apply advanced graph analysis techniques, such as community detection, centrality measures, and resilience analysis, to uncover structural patterns, identify critical components, and assess the potential impact of vulnerabilities on the overall functionality and security of the DApp. Through a series of targeted analyses, we aim to provide new insights into the architectural weaknesses of DApps and contribute to the development of more robust and secure decentralized applications.

We aim to answer the following research questions:

- *RQ1*: Can we identify common development patterns and best practices across networks of contracts and functions in different dApps?
- *RQ2*: How do topological properties of the functions' interaction graph relate to the security risks of DApps?
- *RQ3*: How resilient are DApps to targeted attacks on their function networks, and what are the implications for their overall functionality and security?

In particular, we will show the following results:

- We find common practices concerning the implemented architecture of interactions between functions and contracts across blockchain ecosystems and development teams. This finding suggests the emergence of best practices and inherent constraints in smart contract design, which is valuable for understanding the evolution of DApps' development.
- We identify critical components of the DApp, more likely to be susceptible to technical vulnerabilities. We will show that high betweenness functions act as crucial bridges between different parts of the DApp, making them potential targets for attacks. This insight is important for developers and auditors to prioritize security efforts.
- We demonstrate how the topological properties of function interaction graphs relate to security risks in DApps. Specifically, we show that the small-world nature of these networks facilitates rapid information diffusion, which can be both beneficial for functionality and problematic for vulnerability propagation.
- We quantify the resilience of DApps to targeted attacks on their function networks and discuss the implications for overall functionality and security. Our analysis reveals that removing just 2% of high betweenness nodes can often lead to network fragmentation, highlighting a critical vulnerability in DApp design and providing a quantifiable threshold for network vulnerability.

These results provide a novel perspective on DApp architecture and security: by mapping network properties to potential security issues, we bridge the gap between structural analysis and practical security concerns in DApp development. This approach complements existing smart contract analysis techniques and contributes to ensuring a safer ecosystem for end-users. These results can be used to inform development guidelines and active monitoring, ensuring a safer ecosystem for end-users. Indeed, recently regulators have been looking more closely at ways to tackle and minimise malicious activities in crypto markets [9], and businesses have joined forces to put forward best practices to ensure an increased trust in the technology and support adoption.

To facilitate reproducibility and further research, we have made the dataset used in this study publicly available at https://zenodo.org/records/12731531 [10] and https://zenodo.org/records/13772792 [11].

In the following, we will focus our analysis on 66 DApps written in Solidity, a widely adopted, high-level programming language specifically designed for writing smart contracts on blockchain platforms (e.g., Ethereum). In Sect. 2, we discuss related and complementary literature. The dataset and methodology will be discussed in details in Sect. 3. Finally, we will present the main results in Sect. 4, and we will discuss them in Sect. 5 pointing to future research directions.

2 Related works

Complex systems approaches have shed light on the users' interactions [12], platform's growth, evolution and resilience [13], and market dynamics of crypto ecosystems [14]. In the context of blockchain open-source development, the interplay between developers' team interactions on Github and market behavior of associated cryptocurrencies has been explored [15, 16], highlighting a strong inter-dependence between the code development and assets' valuation. More specifically, complex networks approaches have also been used to analyse blockchain transactions and addresses interactions, to characterise users' behaviour [17], track malicious activities [18], and identify links with cryptocurrency price dynamics [19].

Within the software engineering community, complex network tools have been increasingly used to analyse characteristics of the underlying code structure. The most common approach assumes that software modules are represented as nodes, while relations among them correspond to edges. Other software artifacts, but also people involved in the software development process, have been considered as nodes leading to different kinds of networks. Modeling software systems as networks enabled a graph-based treatment and analysis with the goal of investigating several properties, such as scale-freeness, and the presence of small-world phenomena [20–23]. Object-oriented designs in particular, can be naturally represented as graphs [24]. Software is built up out of many interacting units and subsystems at many levels of granularity (subroutines, classes, source files, libraries, etc.), and the interactions and collaborations of those pieces can be used to define networks or graphs that form a description of a system [25]. In addition, software code remains predominantly a handmade product, produced by human developers, and as such, it is prone to error. The result of a developer error can be directly translated into faults in code and as the world demands ever larger and more complex software systems, controlling faults in code becomes more difficult but increasingly necessary. Understanding fault insertion and fault fixing is crucial to enabling the effective reduction of faults in software systems [26].

In the context of blockchain systems, understanding the network interactions within and among smart contracts could provide new perspectives on system vulnerabilities and operational efficiencies [27]. Recently, researchers have started looking at defining rules and metrics to evaluate smart contract code specifically, within the realm of the so-called blockchain-oriented software engineering research [28]. A number of tools have been developed to analyse code and detect known and typical vulnerabilities, such as *Mythril* and *Osiris* for a smart contract static analysis, *Maian* that detects smart contract vulnerabilities by using dynamic analysis, and *Gasper* used to monitor the gas consumption of smart contracts [2]. Preliminary classifications of typical smart contracts vulnerabilities, such as re-entrancy, computational complexity and overflow have also been conducted [6, 8]. In a recent work, Ibba et al. [29] examines software metrics in DApps to analyse their structural and behavioral characteristics as they grow in complexity. However, to the best of our knowledge, there is limited research on the applicability of complex network theory to the analysis of smart contracts' and DApps' code structure.

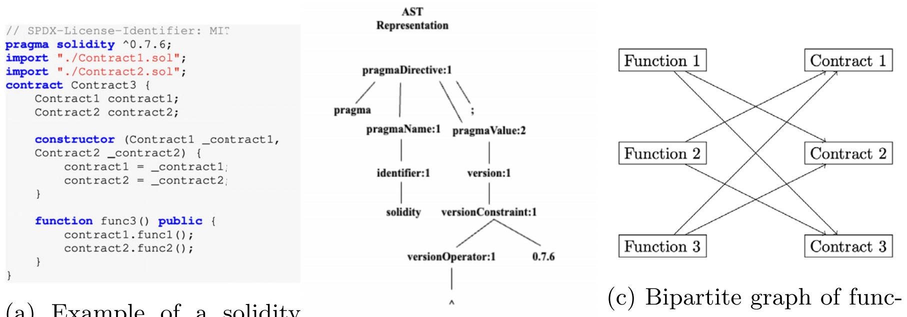
In this work, we aim to bridge this gap by proposing a *complex networks driven software engineering* approach. The DApp's underlying code structure is represented and analysed as a network, whose nodes are functions and contracts, and links represent the strength of the interactions between them.

3 Dataset and methods

Solidity is a high-level programming language specifically designed for writing smart contracts on blockchain platforms [1]. It incorporates elements of pre-existing languages such as JavaScript and Python, but is tailored to the requirements of blockchain development. One of its standout features is its contract-oriented design, which allows for the construction of modular and reusable code structures. This enables developers to create decentralised applications, capable of reproducing complex real-world processes. Analysing Solidity smart contracts is of paramount importance – given its widespread adoption – to assess two critical aspects: platforms' security and robustness of the structural design of DApps. Security vulnerabilities in smart contracts can be dangerous [28], given the immutable nature of blockchain, while the study of Solidity contracts and functions interactions allows to investigate the architecture and operational logic underlying DApps. Indeed, contracts contain the rules and functions that dictate the behaviour of a DApp, making their analysis crucial for understanding how these decentralised systems function. In the following sections we introduce the main steps to gather the data – together with summary statistics and qualitative analysis of the data – and construct contract and function networks.

3.1 Data extraction and parsing

In this work, we focus on a dataset composed of DApps mainly supported by the Ethereum blockchain, but including also examples from other blockchains as Binance, Optimism, Polygon, Astar, Shiden and ThunderCore. The data on the underlying smart contract code is obtained from the Github repository of each DApp. For each DApp, the associated smart contracts code is broken down into relevant sub-components (e.g., libraries, functions, etc.) using an *ad hoc* tool specifically built to recognize these sub-parts in Solidity contracts [30]. More specifically, we use the tool *MindTheDApp*, designed for the structural analysis of DApps built with Solidity contracts [30]. The tool uses ANTLR4 [31] to traverse the Abstract Syntax Tree (AST) – a tree representation of the abstract syntactic structure of the source code – of Solidity contracts. ANTLR4 works by accepting grammar rules to automatically produce both a lexer and a parser. The lexer first breaks down the input Solidity code into tokens, eliminating unnecessary elements such as whitespaces and comments. These tokens are then processed by the parser to form an Abstract Syntax Tree (AST), which organizes the code into a hierarchical structure useful for the analysis. The data extraction procedure using the tool *MindTheDApp* is schematically described in Fig. 3.



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;
import "./Contract1.sol";
import "./Contract2.sol";
contract Contract3 {
    Contract1 contract1;
    Contract2 contract2;

    constructor (Contract1 _contract1,
    Contract2 _contract2) {
        contract1 = _contract1;
        contract2 = _contract2;
    }

    function func3() public {
        contract1.func1();
        contract2.func2();
    }
}
```

(a) Example of a solidity contract.

(b) Abstract Syntax Tree (AST).

(c) Bipartite graph of functions and contracts.

Figure 3 The goal of the tool *MindTheDapp* is to create a bipartite graph that illustrates the interactions between functions and contracts. Combining the ANTLR4 and Abstract Syntax Tree (AST) modules, the tool extracts various elements including contract calls, modifiers, constructors, and events. The Figures above shown are introduced in the original paper by Ibba et al. [30]. Figure 3a: Example of a solidity contract. Figure 3b: After receiving the Solidity source code, ANTLR4's lexer first tokenizes the input, breaking it down into identifiable lexical units. These tokens are then fed to ANTLR4's parser, which organises them into a hierarchical structure, resulting in an Abstract Syntax Tree (AST). Figure 3c: Once the AST is constructed, the Analyzer module scans and extracts key elements, such as contracts, functions, interfaces, events, modifiers, and libraries. In the resulting bipartite graph, nodes represent two distinct categories: the first category comprises functions, events, and modifiers, while the second category includes smart contracts, interfaces, and libraries. An edge exists between a node from the first category and a node from the second category if and only if the function, event, or modifier from the first category calls the corresponding contract, interface, or library in the second category

Table 1 Example of dataset returned by the tool for the DApp *Aave* (Ethereum - DeFi)

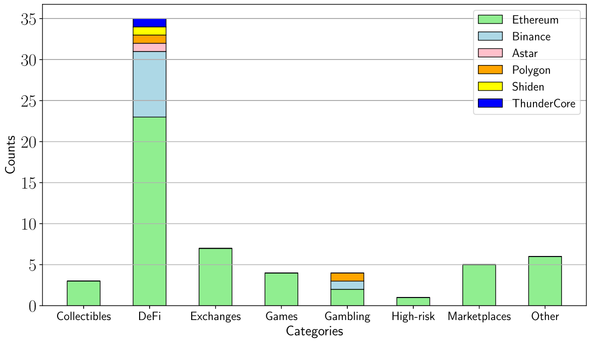
File	Source_Contract	Source_Function	Target_Contract
WETH9Mock.sol	WETH9Mock	mint	WETH9Mock
WETH9Mock.sol	WETH9Mock	mint	None
MockBadTransferStrategy.sol	MockBadTransferStrategy	constructor	MockBadTransferStrategy
MockBadTransferStrategy.sol	MockBadTransferStrategy	performTransfer	MockBadTransferStrategy
...	...	...	...

3.2 Dataset features

For each DApp the tool gives as output a CSV file, containing information regarding functions invoking contracts, allowing for advanced network analysis. The dataset obtained from our parser comprises for each DApp the information on the File in which the contract is defined, on the Source Function, i.e. the function calling a target contract (for example Function 1 in Fig. 1), the Source and Target Contract, respectively the contract which the function belongs to and the contract the function is called by (Contract A and B in Fig. 1). In Table 1 we provide an example of the parser output for *Aave* (category: Ethereum - DeFi; balance: \$108.85B; ranking: #2 in DeFi, #56 in General). As a lending protocol, *Aave* allows users to supply assets and earn passive income.

Overall, our dataset consists of 51 DApps Ethereum-based, and 15 DApps deployed on other blockchains (see Fig. 4 and Appendix A). The majority of DApps are Ethereum-based, due to the significant expansion of the Ethereum ecosystems in recent years [32]. These applications are related to multiple sectors, such as:

DeFi Applications in this category handle various aspects of financial services: Insurance, Investments, Lending and Borrowing, Payments, Token Swap, and Trading and Pre-



(a) Composition of the dataset in terms of DApp's category/use-case and compatible blockchain.

	No. of DApps
Ethereum	51
Binance	10
Astar	1
Polygon	2
Shiden	1
ThunderCore	1

(b) Number of DApps by blockchain protocol.

Figure 4 Overview of the dataset

diction Market. Each of these sub-categories brings a unique set of functionalities, all aiming to disrupt traditional financial systems by introducing automation, transparency, and efficiency through blockchain and smart contract technology.

Art and Collectibles This category of DApps focuses on digital ownership and artistic creation. Tokenization, based on so-called Non-Fungible Tokens (NFTs), plays a critical role in use cases related to establishing ownership and provenance.

Gaming These DApps offer interactive entertainment and virtual exploration, and they are generally divided into Competition and Digital World sub-groups. They allow buying and trading digital assets that can enhance gameplay, and they operate in environments that simulate various landscapes.

Technology This category contains DApps that aim to revolutionize developers' tools and integrate blockchain into existing technology platforms. They support open-source development initiatives and facilitate the decentralization of various technological services.

Gambling Gambling DApps comprise platforms allowing users to bet their money on gambling and high-risk games. They range from decentralized casinos to prediction markets.

Staking Staking DApps are decentralized applications that allow users to lock their cryptocurrencies to support network operations, often in exchange for rewards or other benefits. It offers a way for users to potentially earn returns on their crypto holdings by participating in network security or governance.

We decided to concentrate on DeFi-related DApps as, in 2021, DeFi protocols emerged as the predominant targets of cryptocurrency hackers, and this pattern further intensified in 2022 [33].

The number of Source Contracts that compose a DApp is taken as a proxy for its size, following [29]: DApps are categorized into Small (3 to 23 contracts), Medium (24 to 45 contracts), and Large (46 to 193 contracts). The categorization of DApps into Small, Medium, and Large groups – based on the number of contracts they are formed of – is a heuristic approach driven by the characteristics of our dataset. The specific ranges (3 to 23, 24 to 45, 46 to 193) for these categories were selected to create a balanced sub-division that allows meaningful comparison and analysis across groups. We, therefore, have 20 Small DApps, 22 Medium DApps, 22 Large DApps (see Fig. 5). This metric offers a quantitative measure

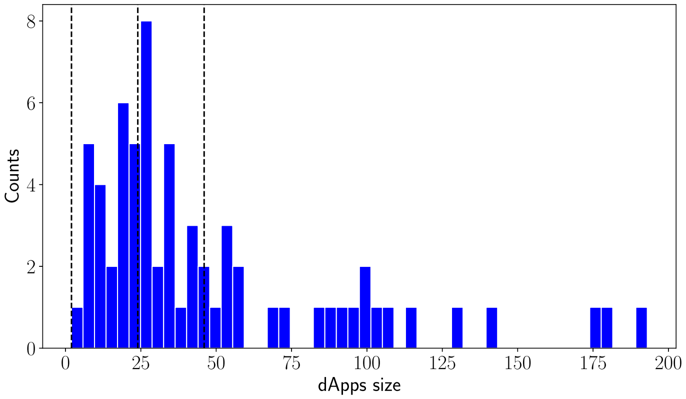


Figure 5 Distribution of DApp sizes. The number of *Source Contracts* is a proxy for the size. Our dataset is composed of 20 Small DApps, 22 Medium DApps, and 22 Large DApps

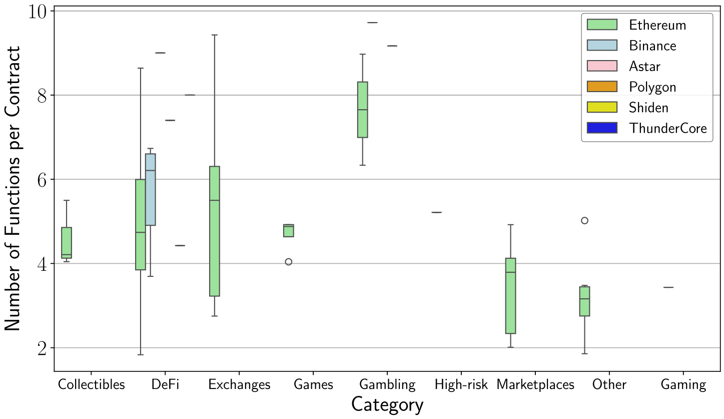


Figure 6 Box plot of the number of functions per contract. The ends of the box represent the first and third quartiles, the median (second quartile) is marked by a line inside the box, and the end of the whiskers represent the minimum and the maximum

of a DApp's complexity and potentially its functional diversity. The smallest DApp is *1inch Network* (Ethereum - DeFi) with 6 functions and 2 contracts, while the largest is *Balancer* (Ethereum - Exchanges) with 531 functions and 193 contracts.

Functions are the fundamental building blocks of contracts and, therefore, DApps. In Fig. 6, we analyse the number of functions in each DApp, adjusted for their respective sizes. On average, the number of functions is 5.09 times the number of contracts within the same DApp, with a standard deviation of 1.91. The minimum value is 1.83 for the Ethereum - DeFi category, while the maximum value is 9.43 for the Ethereum - Exchanges sector. The DeFi and Exchanges categories show the highest dispersion, with values exhibiting significant deviations from the median. The Gambling category is, instead, characterised by a larger number of functions on average.

Several categories exhibit very narrow distributions, indicating low variability in the number of functions per contract. For instance, the Collectibles category shows a tightly clustered range of functions per contract, suggesting a more standardized approach in how these contracts are structured. This could be related to the increased dominance of standardized protocols such as ERC-721 and ERC-1155 in the Ethereum ecosystem (and similarly in other blockchain protocols) has led to a more uniform structure of collectible contracts. These standards define a set of core functions and interfaces that are consistently implemented across different collectible DApps, resulting in a reduced need for additional, unique functions per contract. The median number of functions per contract varies across categories. For example, the DeFi category, which is more variable, still shows a relatively high median, indicating that DeFi contracts tend to have more functions compared to categories like Collectibles or Games. Categories such as Exchanges and Gambling display significant variability with some outliers, indicating that while many contracts are standardized, there are a few with a significantly different number of functions. This could reflect experimental or customized implementations within these categories. We hypothesize that the complexity of the dApp is related to its use case, community interest, and investments. DeFi and Exchanges are indeed the sectors with more experimentation, and associated protocols record the highest number of transactions according to recent experimental studies [34]. The high function complexity in DeFi and Exchanges therefore aligns with their popularity and the substantial investments they attract. The robust community interest and financial backing encourage developers to explore diverse functionalities, leading to more sophisticated and feature-rich DApps. The correlation between high transaction volumes and function complexity supports this hypothesis, as DApps in these sectors must manage numerous and varied operations efficiently.

The presence of numerous functions suggests the reliance on multiple separate scripts to accomplish different tasks. This technique of splitting larger tasks into multiple subfunctions suggests a reduction in the responsibility of single contracts. If all tasks were concentrated in few functions, the likelihood of the DApp ceasing to function in the event of technical malfunctions would be considerably higher. Instead, by distributing tasks across a greater number of functions, the risk of a technical malfunction affecting the entire DApp is minimized. This result is not surprising, as a similar approach is observed in 'standard' (i.e., non blockchain-related) software engineering. In software engineering, the principal mechanism employed for designing object-oriented software is the class. The allocation of responsibilities and collaborations among classes can take various forms. In a delegated control style, a well-defined set of responsibilities is spread across multiple classes. These classes assume distinct roles and occupy recognized positions within the application architecture. Object-oriented design experts suggest that a delegated control style is more comprehensible and adaptable than a centralized control style [35]. This approach shares similarities with our findings in the context of DApps, where a distributed approach to functions mitigates the impact of potential technical failures on the entire system.

3.3 Building contract and function networks

As shown in Table 1, the dataset reveals interactions between the calling function and the contracts, which the call originates from and terminates into. We are indeed interested in conducting a more fine grained analysis, as our objective is to understand interactions *within* contracts and interactions *within* functions (considering functions with the same Source Contract as identical). To construct the contracts network for each DApp, we use information on the Source and Target Contract of each function call. We build the network's adjacency matrix, where the rows denote the sources and the columns denote the

Table 2 Example of adjacency matrix for *Aave*'s (Ethereum - DeFi) contracts network

	ACLManager	AToken	ATokenHarness	AaveEcosystemReserveController	...
ACLManager	1	0	0	0	
AToken	0	9	0	0	
ATokenHarness	0	0	1	0	
AaveEcosystemReserveController	0	0	0	5	
...					

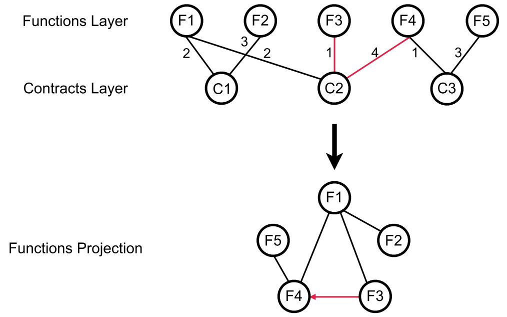


Figure 7 Illustration of the bipartite network Functions - Contracts, and its projection on the Function layer. For example, the edge weight from node *F*3 to *F*4 is computed as follows: it is equal to 1 (the number of possible directions when starting from node 3, weighted by the probability of going through that link) multiplied by the probability of reaching *F*4, divided by the total number of nodes through which the information can flow (weighted by their respective probabilities): $1 \cdot \frac{4}{2+1+4} = 0.57$

targets (Table 2). The matrix element at position *(i, j)* may assume a value of 0, if the function belonging to the Source Contract *i* does not call Target Contract *j*, or it may assume a value of $n \in \mathbb{N}$, if the function in Source Contract *i* calls Target Contract *j* *n* times. We, thus, obtain for each DApp a weighted directed network of contracts interactions, resulting in 66 contracts networks.

To infer the network of connections among functions, further steps are necessary. We use the information regarding the Source Function and the Target Contract: analysing the relationship between them is crucial to determine the system's robustness. Indeed, vulnerabilities in function-contract calls have been exploited in hacking attacks aimed for instance at stealing funds from cryptocurrency wallets and applications [28, 36]. As previously done, we build the bi-adjacency matrix, where the rows denote the Source Functions and the columns denote the Target Contracts. The matrix element at position *(i, j)* may assume a value of 0, if function *i* does not call target contract *j*, or a value of *n*, if function *i* calls target contract *j* *n* times. Given the possibility of multiple calls from the same function to the same contract, we obtain a weighted bipartite graph. The two layers are Functions (layer *F*) and Contracts (layer *C*) as schematically depicted in Fig. 7, top panel. Since our interest lies in the relationships *within* the functions layer, we project the information onto the single layer *F*. The one-mode projection onto layer *F* results in a network consisting exclusively of *F* nodes, and it is a procedure extensively used in graph theory. Determining how to weight the edges in this network is a critical aspect of the one-mode projection. We adopt a methodology similar to the one introduced by Tao Zhou et al. in [37]. In order to assess if the contract $c \in C$ called by function $f_1 \in F$ is more likely to be called also by function $f_2 \in F$ we have to perform a contraction of the bi-adjacency matrix *M* over the contract dimension, i.e., the set *C* of contracts, and take the element (f_1, f_2).

Table 3 Example of bi-adjacency matrix for *Aave*'s (Ethereum - DeFi) functions network

	_approveDelegation	_approve	_approve_Incentivized	_burnScaled	...
_approveDelegation	0.009	0.009	0.009	0.03	
_approve	0.009	0	0.06	0.03	
_approve_Incentivized	0	0	0	0.11	
_burnScaled	0.004	0.004	0.004	0.018	
...					

This method is called *probabilistic spreading approach*. Let us consider one bit of information on a generic function $f_1 \in F$. We aim to describe how this information can spread to contracts in C, then back to F. Firstly, the information moves to the contracts layer according to the connection patterns of M. The probability that the information goes from f_1 to a given contract c is

$$\rho_{f_1 \to c}^{F \to C} = \frac{M_{f_1,c}}{\sum_{\tilde{c} \in C} M_{f_1,\tilde{c}}}, \tag{1}$$

where $\sum_{\tilde{c} \in C} M_{f_1,\tilde{c}}$ is the number of possible paths from f_1 to C, each weighted by the probability of going through a given path. Since the elements of $M = 0, 1, \dots, n$ we are not assuming equal transition probabilities, introducing a bias in the process. Secondly, the information that reached the contracts layer jumps back to the functions one, following again the connection patterns of M. The transition probability from c to a given function f_2 in layer F is:

$$\rho_{c \to f_2}^{C \to F} = \frac{M_{f_2,c}}{\sum_{\tilde{f} \in F} M_{\tilde{f},c}}, \tag{2}$$

where $\sum_{\tilde{f} \in F} M_{\tilde{f},c}$ is the number of possible weighted paths from c in layer C to layer F. Finally combining these steps, the probability that the bit of information jumps from function $f_1 \in F$ to function $f_2 \in F$, via all possible connected contracts c, is

$$\rho_{f_1 \to f_2}^{F \to F} = \sum_{c \in C} \rho_{f_1 \to c}^{F \to C} \rho_{c \to f_2}^{C \to F} = \frac{M_{f_1,c}}{\sum_{\tilde{c} \in C} M_{f_1,\tilde{c}}} \frac{M_{f_2,c}}{\sum_{\tilde{f} \in F} M_{\tilde{f},c}}. \tag{3}$$

Equation (3) defines a monopartite network of F nodes, which can be interpreted as the flow of information *within* functions in F. We can interpret the connections of this network as conditional probabilities $P(f_2|f_1) = \sum_{c \in C} P(f_2|c)P(c|f_1)$.

We thus obtain for each DApp a weighted directed monopartite network of functions interactions, resulting in 66 functions networks. In Table 3 an example of the bi-adjacency matrix for *Aave*'s functions network.

To assess the statistical significance of the elements of the matrices defined in (3), we resort to a null model. We use the disparity filter,[1] a filtering method that extract the relevant connection backbone in complex networks, preserving the edges that represent statistically significant deviations with respect to a null model for the local assignment of weights to edges [38]. An important aspect of this method is that it does not affect small-scale in-

[1] https://github.com/DerwenAI/disparity_filter.

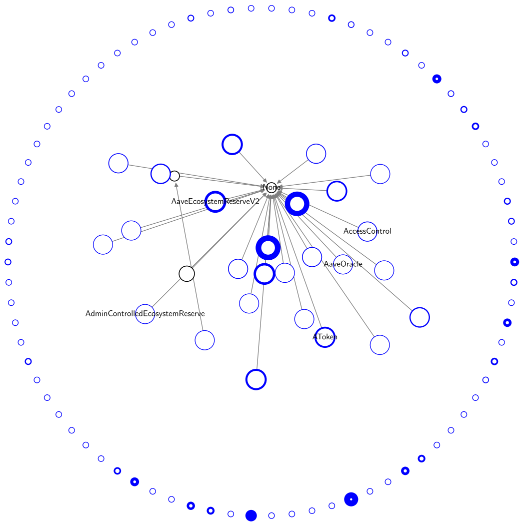


Figure 8 Network of contracts' interactions for *Aave*. In blue, the nodes with self-loops. The width of the border of each blue node corresponds to the weight of its self-loop link. The nodes positioned on the external circle do not interact with other contracts

teractions and operates at all scales defined by the weight distribution. In this context, the information on weights is significant as they directly correlate with the frequency of functions calling a contract, influencing the associated gas fee expenses. We adopt a filtering method that retains edges that represent statistically significant deviations when compared to a null model of local weight assignment. It filters out connections characterized by substantial disorder, while preserving structural properties and hierarchies. Our findings from the network analysis indeed reflect intrinsic characteristics of the systems we are examining, rather than being a mere consequence of the chosen filtering method. In Sect. 4, we report the results of the analysis conducted on the filtered weighted functions and contracts networks.

4 Results

We analyse 51 Ethereum-based DApps and 15 DApps from other blockchains (Binance, Optimism, Polygon, Astar, Shiden, and ThunderCore), spanning various categories, and of varying sizes. **Contracts Networks** We generate a total of 66 weighted directed networks illustrating contracts' interactions, with each network representing a DApp. In these networks the nodes represent a contract, and the width of the links reflects the strength of interactions from the source to the target contract (how many times it is called), while the node sizes are scaled based on the number of target contracts that a given contract calls. In Fig. 8 the network of contracts' interaction for *Aave* is presented. The names of the contracts with the highest betweenness centrality are listed. For instance, *None* is a *Context* contract, a dependency used to return the contest of transaction sender and data. In Fig. 18 (Appendix B) several other examples of contracts networks are presented. We decide to present the networks of DApps that, at present, exhibit notable balances – defined as the total value of current assets held in the DApp's smart contracts. To ensure a representative sample, we include examples from various categories and blockchain platforms. Figure 9 shows the distribution of networks' degrees and density, and the prevalence of self-loops as the only connections. The networks exhibit a low degree, with the

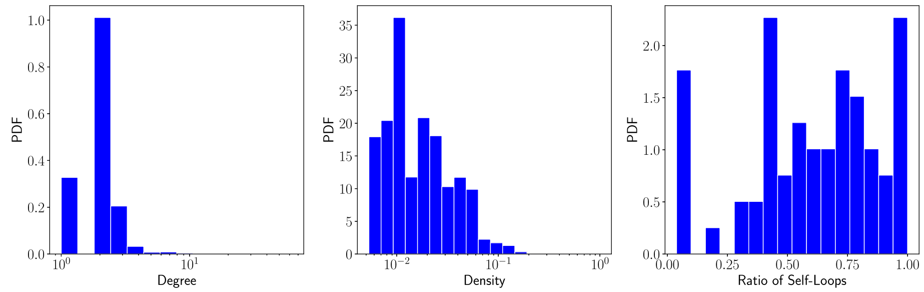


Figure 9 Left Panel: Probability Density Function (PDF) of the degree distribution for the 66 contracts networks. Central Panel: PDF of the density of the networks. Right Panel: For each network, the ratio of nodes having a self-loop as their sole connection to the total number of nodes is presented

majority falling within the range of 2 to 3: the nodes have few connections. The Left and Central panels reveal consistent characteristics within the contracts networks of DApps, regardless of their size or category, specifically their sparsity. The density of a graph is a measure of how many potential edges are present in the graph compared to the total number of possible edges in a complete graph of the same size. The networks display a low density, with the exception of *1inch Network*, which has a density of 1.0. However, as said in Sect. 3, *1inch Network* has only 2 contracts, making this result quite trivial. In the Right panel, it becomes apparent that the majority of links consists of self-loops. A total of 54 networks have a minimum of 40% of nodes with self-loops, and 33 networks (more than half of the dataset) have at least 60% of their nodes connected solely through self-loops. A Louvain modularity analysis on the undirected version of these graphs produces an average modularity coefficient of 0.8 across all networks. In conclusion, the networks exhibit evident sparsity, with the majority of connections being self-loops, and community structure is highly significant, resulting in a lot of distinct components.

In a software engineering framework, it means that the DApps are designed with a high level of independence and minimal inter-contract dependencies. This choice may be a deliberate strategy to improve security and reduce the risk of chain failure, given the immutability of contracts when deployed. The presence of self-loops indicates that most contracts are self-sufficient, executing functions and maintaining a state without the need for external calls or interactions. Instead, the presence of few communities indicates sets of contracts grouped by functionality or purpose, facilitating maintainability and potential scalability.

Functions Networks We generate a total of 66 weighted directed networks illustrating functions' interactions, with each network corresponding to a specific DApp. On average, the ratio between post-filter and pre-filter nodes stands at 65%, with a standard deviation of 14%. The minimum ratio of 32% is observed in the case of *SWAPP Protocol* (Ethereum - DeFi), which features 214 nodes pre-filter (i.e. functions) reduced to 71 nodes post-filter. In contrast, *Plexus* (Ethereum - Exchanges) shows the maximum ratio of 97%, having initially 66 nodes, which are reduced to 64 post-filter. The networks are visualized using the *spring* layout; the nodes represent functions, and the width of the links reflects the strength of interactions between functions. The node sizes are adjusted according to the number of target contracts called by each function, and nodes are displayed in a purple shade if they rank among the nodes with the highest betweenness, while a green color is assigned to

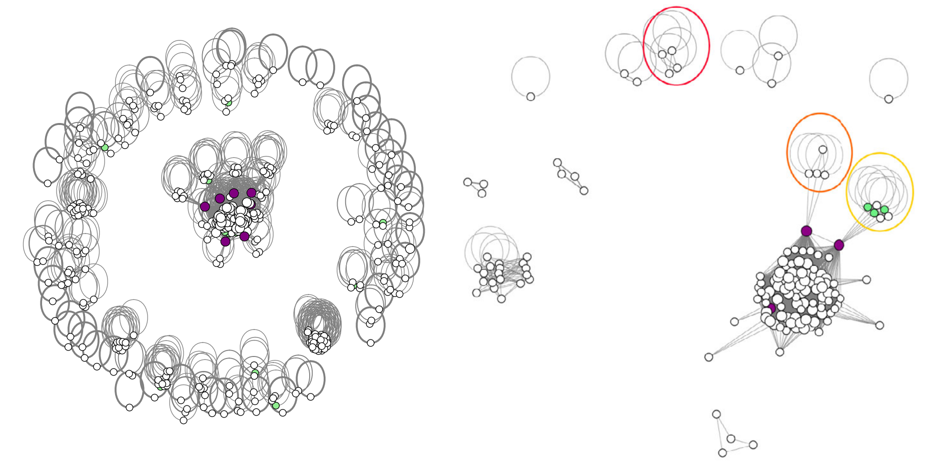

(a) Network for functions interactions pre-filter (b) Network for functions interactions post-filter

Figure 10 *Aave* Functions Network (Ethereum - DeFi). In purple the nodes with highest betweenness (i.e. *cancel*, *withdrawFromStream*, *setClaimer*), in green the nodes with highest clustering coefficient (i.e. *executeDelegateCall*, *updateDelay*, *updateGracePeriod*). The purple functions are responsible for authorizing an address to withdraw tokens on behalf of another specific address, clearing a queue of actions for execution, or making a token withdrawal from a Stream. The green functions are responsible for updating the time (setting the value to the next moment when the execution of a set of actions concludes) and adjusting the delay between queuing a set of actions and their execution

the nodes with the highest clustering coefficient. In Fig. 10 the network of functions interactions for *Aave* is presented. We observe a set of smaller components that represent the secondary functionalities of the DApp. The characteristic of the contract network having a high number of self-loops translates into this network as all minor components consisting of functions defined within the same contract. For instance, in the red component, there are only functions defined within the *PolygonBridgeExecutor* contract. In the orange community functions are defined within *AaveEcosystemReserveV2*, and in the yellow community they belong to *BridgeExecutorBase*. In contrast, the core of the largest component comprises functions defined in multiple contracts, representing the main functionality of the DApp. For instance, within it, we find functions such as *setReserveInterestRate*, *setPoolImpl*, and *setLiquidationProtocolFee*, of fundamental importance for the functioning of the DApp.

These functions are integral as they handle critical operations such as managing interest rates, updating the pool implementation, and setting liquidation protocols. This example highlight how different communities within the functions' network shape specific sets of higher-level functionalities. The smallest components typically focus on specialized tasks confined within a *single contract*, while the largest component integrates functionalities *across multiple contracts*, showing the collaborative nature of core operations in the DApp. In Fig. 19, 20, 21 (see Appendix C) several other examples of these network visualizations are presented. The structure is consistent across all of them: there is always a largest component for the main functionality and a series of minor ones.

4.1 Characteristics of networks of functions in DApps

In the case of functions networks, Fig. 11, still reveals consistent characteristics across DApps, regardless of their size or category, but the scenario differs from what was observed

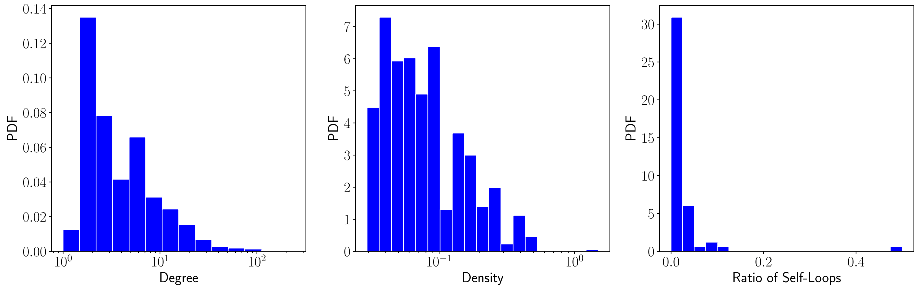


Figure 11 Left: PDF of the degree distribution for the 66 functions networks. Central: PDF of the density distribution. Right: For each network, the ratio of nodes having a self-loop as their sole connection to the total number of nodes

previously with contracts. The degree of the functions networks is higher compared to the case of contracts networks, signifying a greater degree of connection. Similarly, the network density is higher, suggesting a lower sparsity in the graphs. In the case of *1Inch Network*, the density is 1.6, as self-loops are included in the total count of edges, which can result in densities exceeding 1. Finally, the proportion of nodes having self-loops as their sole connection is significantly smaller compared to contracts networks: out of the 66 networks, 65 have a fraction of nodes connected solely by self-loops that accounts for 15% of the total nodes. On average, an analysis of Louvain modularity within the networks gives a result of 0.69, confirming reduced division into distinct components and highly connected nodes. This suggests greater complexity in interactions and a greater level of integration and task sharing among functions within the same DApp.

Networks' metrics and typical patterns in DApps In Fig. 12 - top panel, we show the relationship between the diameter of the largest connected component and the number of components in DeFi DApps is presented. Similar results and plots are for non DeFi DApps in Fig. 22 in Appendix D. Our emphasis is placed on the largest component as it represents the most crucial part of the network, housing the core functionalities of the DApp. Functions outside of this component perform less essential actions. The plots reveal consistent trends regardless of the DApp's category, affirming the presence of a common development pattern that is independent of the DApp's intended purpose. As previously mentioned, a distinct division into separate components is evident (with the number of components obviously increasing with the number of functions within the network). Functions within these DApps tend to form discrete groups with limited interactions between these groups, indicating a certain degree of compartimentalization in how functions are structured and interact within the application. Different sets of functions perform specific roles and maintain limited direct interactions with functions defined in other contracts. Additionally, as DApps increase in complexity by incorporating more functions (as indicated by the larger data points on the scatter plot), their internal network structure appears to become more intricate. The maximum number of connections within the largest component tends to grow with the number of functions in the DApp. Larger DApps may exhibit greater specialization, requiring a broader range of functions to manage specific tasks and fostering increased interaction and communication among these functions. The clustering coefficient is the fraction of all possible pairs of neighbors of node *i* that are themselves linked in the graph. The clustering is like a local version of the betweenness, which is in

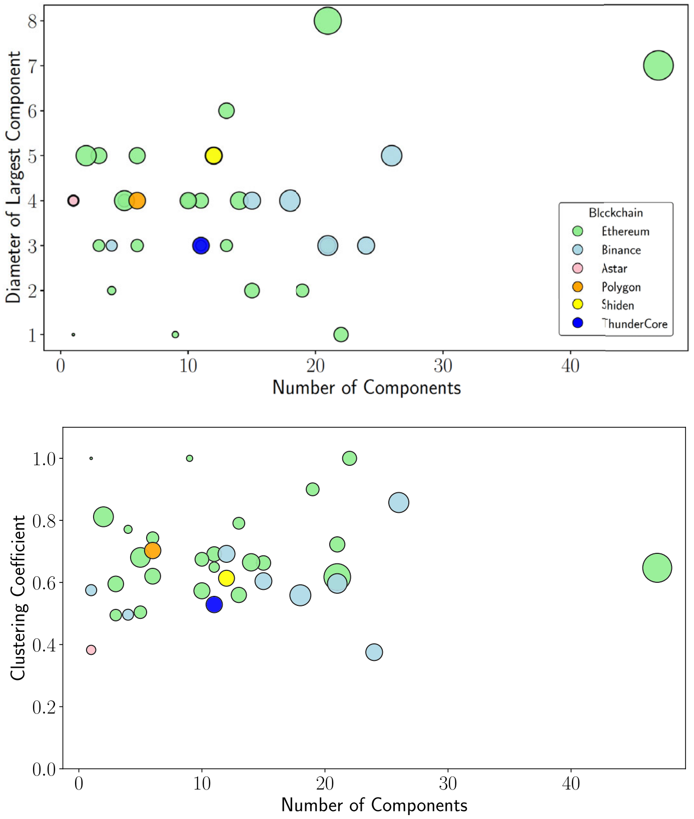


Figure 12 Top: Scatter plot of the diameter of the largest connected component as a function of the number of components. Bottom: Scatter plot of the global clustering coefficient in the largest components vs the number of components. Each dot represents a DApp, and the size of the dot is proportional to the number of functions in the specific DApp. This plot is restricted to DApps belonging to the DeFi category

turn a measure of centrality based on shortest paths. Betweenness and local clustering are, indeed, correlated [39]. If a vertex has a larger local clustering value, then the neighbors of the vertex can directly communicate with each other rather than going through the particular vertex. If the neighbors of a vertex do not need go through the vertex for shortest path communication, then it is more likely that the rest of the vertices in the network would not need to go through the vertex for shortest path communication. If a vertex has a smaller local coefficient, then the neighbors of the vertex are more likely to go through the vertex for shortest path communication between themselves (as there is more likely not a direct edge between the two neighbors, because of the low local coefficient for the vertex) [40]. Therefore, we expect an inverse relationship between these two measures in our networks. In Fig. 13, we present an analysis of the relationship between betweenness and clustering coefficient of each node located within the largest component of each function network. Our goal is to discern whether there exists a distinctive characteristic specific to the DApps. Given the consistent network patterns identified through our prior analyses, regardless of the DApp's category or the blockchain the DApp is deployed on, we choose to analyse all nodes across all function networks together. As expected, the plot exhibits a clear trend, reinforcing a notion of similarity and consistent structural patterns across the function networks. In the Left Panel, we illustrate the relationship between the two quantities in the original networks. In the Right Panel, we

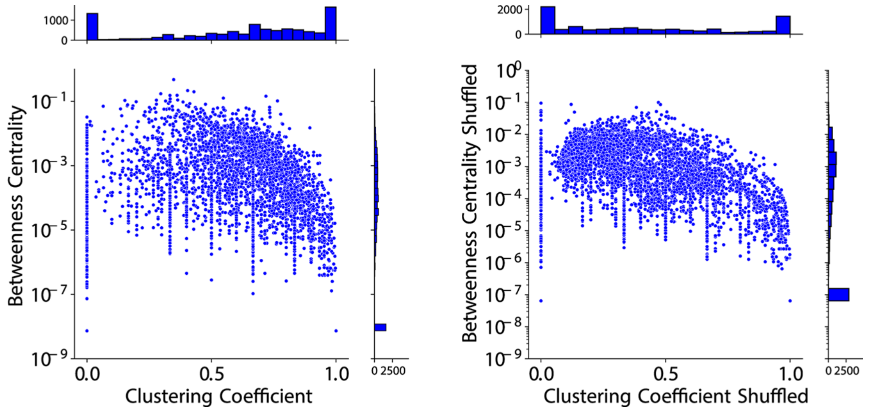


Figure 13 Left: Relationship between clustering coefficient and betweenness in the original networks. Right: Relationship between clustering coefficient and betweenness in a random null model

present the same relationship calculated for a null network model, constructed as configuration model retaining the same number of nodes and block structure while shuffling the links. For each network, we construct a random one, which retains the same modular structure as the original one, but connects nodes within each sub-component randomly. This null model preserves the modularity of the original networks, meaning that the compartimentalization of functions and the connections among functions performing similar tasks are maintained, but randomly reconnect functions designated for a particular task. This ensures that the results we obtain are not a mere consequence of the network's structural characteristics. As expected, Fig. 13 shows that nodes with higher clustering coefficient tend to have lower betweenness centrality. This implies that nodes surrounded by highly connected neighbors, located in densely connected areas of the graph, are the same nodes with limited involvement in shortest paths between other nodes. As a result, they are not essential for the overall network connectivity nor efficient information transmission. Instead, nodes with high betweenness, serving as fundamental intermediaries for the flow of information or influence within the network, work as connectors between highly connected areas. However, when comparing our networks to the null model, we observe significant differences in the distribution of the clustering coefficient. In our networks, the mean clustering coefficient is higher, with the majority of data points falling within the range of 0.5 to 1. In contrast, the null model exhibits data points with lower clustering coefficients. Furthermore, the null model displays lower and less variable betweenness centrality values compared to the original real networks. A schematic representation of the structural differences between the real and randomised version of the networks can be seen in Fig. 14. Therefore, our systems have characteristics that are independent of their modular structure. These characteristics include variable betweenness centrality with lower values and higher clustering coefficients, implying that our networks inherently consist of highly interconnected communities within each component, with only few nodes serving as bridges between them. This analysis helps detect patterns in dApps function networks, highlighting common emergent structural features (e.g., in terms of bridge-functions) previously undetected nor flagged in smart contract analysis, therefore contributing to answering *RQ1*. In Sect. 5.1, we will further discuss the relationship between such structural properties and specific security threats and best practices sugges-

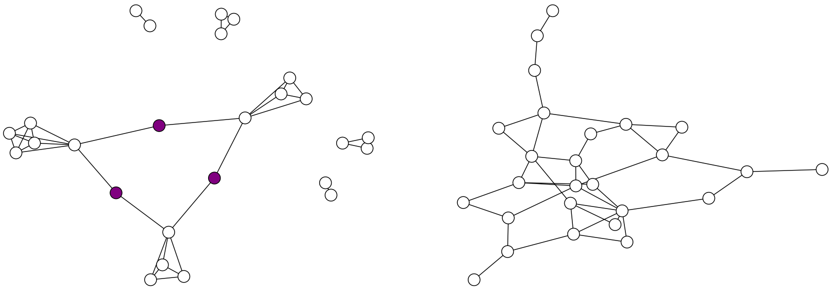


Figure 14 Left: Scheme of the typical real network structure of functions interactions. There is a largest component and few minor disconnected components on the side. The largest component is composed of different communities, with few bridges between them (the purple nodes). These sub-components are complete sub-graphs. Right: Randomised version of the function network with same number of nodes

tions for auditors and development teams In the function network examples in Figs. 19, 20, 21 in Appendix C, nodes with higher betweenness are marked in purple, while those with a higher clustering coefficient are marked in green. The purple nodes act as intermediaries between sub-components.

Interconnectedness of functions In network theory, small-world networks are distinguished from other networks by two properties: high clustering coefficient and short path lengths (as commonly observed in random networks). This network type is known for its ability to support rapid diffusion of information or processes across the network. Even if two nodes may be distant from each other, there are relatively short paths that indirectly link them, enabling quick transmission of information. In Fig. 12 - bottom panel, we present the global clustering coefficients of nodes in the largest components of DeFi DApps. Similar results are obtained for non-DeFi related DApps and are shown in Appendix D in Fig. 22. In Fig. 15 we provide a comparison between the average path lengths in the largest connected components of the real networks and the ones of randomly generated networks with the same number of links and nodes. DApps exhibit high clustering coefficients and low average path lengths, similar to the ones of random networks with same number of nodes and links. The results suggest a similarity with the structure of a small world network, indicating the presence of substantial local interactions, efficient information flow within the component, and connectivity between functions, even when they are not directly linked, allowing fast information diffusion and effective interaction among functions. This result provides insights on structural and topological properties of the function network as per *RQ2*. We will further connect these properties to specific security risks in Sect. 5.1.

Information diffusion In graph theory, the concept of a clique is fundamental to understand the connectivity of networks. A clique is a group of vertices within a graph where each vertex is directly connected to every other vertex in the group, and its size is the number of vertices it contains. A maximal clique is a clique that cannot be extended by including one more adjacent vertex, meaning it is not a subset of a larger clique. Figure 16 shows

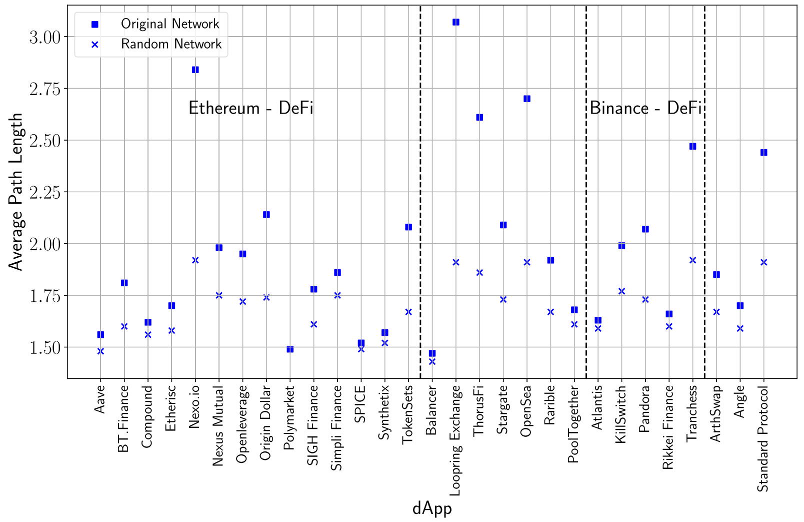


Figure 15 The plot shows a comparison between the average path length in the largest connected component of the real (■) and random network (×). We consider only the networks where the largest connected component contains more than 50 nodes

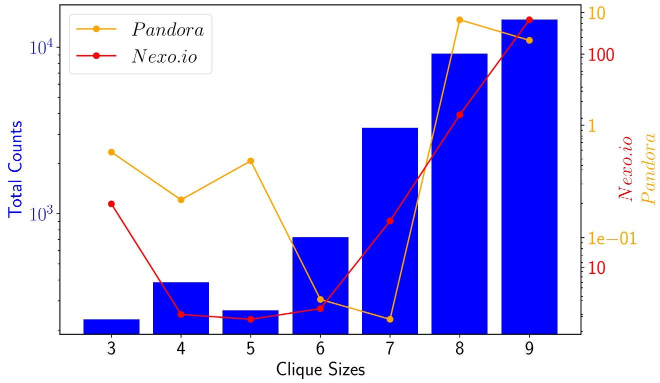
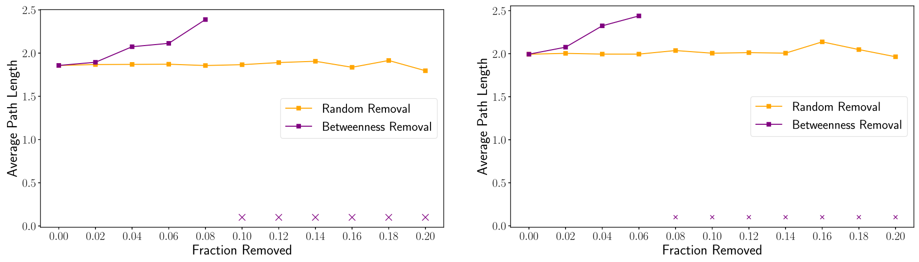


Figure 16 Maximal cliques' sizes across all networks. For each network, we analyse the list of the maximal cliques and normalize them by the size of the respective DApp. The cumulative cliques' sizes across all networks are shown in blue. There are two examples overlaying the bar chart: in orange, *Pandora* (Binance - DeFi, 40 contracts), a medium-sized DApp, and in red, *Nexo.io* (Ethereum - DeFi, 176 contracts), a large DApp

the distribution of these maximal cliques of dimension 3 to 9 across the DApps networks, revealing a consistent trend across different sizes. The pattern shows the prevalence of large maximal cliques, meaning that the information on a function is just one step away from another, so the information diffusion process is immediate. The observed (large) size of the maximal cliques points to the fact that dApps are highly integrated systems, where the data flow between functions is fast, with a continuous exchange of processed inputs and outputs between functions. This result replies to *RQ2*.

4.2 Functions network resilience to targeted attacks

We can further analyse the largest connected components' resilience by examining the behaviour of average path length in the largest connected component in different conditions. In this analysis, we only consider networks with more than 50 nodes in the largest



(a) *Simpli Finance* (Ethereum - DeFi) (b) *KillSwitch* (Binance - DeFi)

Figure 17 The plot shows the average path length within the largest connected component as a function of the fraction of nodes removed within the range of 0% to 20%. In orange, it represents the change in the average path length when nodes are randomly removed, while in purple, it illustrates the effect of removing nodes with the highest betweenness centrality, starting from the node with the highest betweenness and proceeding accordingly. The betweenness centrality calculation was performed at the beginning of the component analysis and it is not recomputed after each node removal. When the component becomes disconnected, as observed in the case of *Simpli Finance* when more than 8% of nodes are removed, a cross is marked on the plot

component. In Fig. 17, we investigate how the average path length changes as an increasing fraction of nodes is removed, ranging from 0% to 20%. With targeted removal, the distances between the remaining nodes tend to increase, leading to the fragmentation into smaller, disconnected components. Targeting nodes with high betweenness centrality within DApps results in the complete disconnection of the component, a phenomenon not arising from the random node removal. The cases presented are the scenarios where the targeted removal of nodes keeps the component connected for as long as possible. However, in the majority of cases (20 DApps out of 29), the component becomes disconnected after the removal of just 2% of the nodes [2]. In the event of a hacking attack, the DApps represented in the Fig. 17 are the only ones that maintain uninterrupted information flow for a longer period, even when specific functions cease to operate. For all the other DApps, it is evident that an attack on a small percentage of those functions characterised by the highest betweenness centrality (indicating the presence of significant information flow pathways), has the potential to disrupt the DApp's functionality. This analysis direclty informs on replies to resilience of DApps to targeted attacks as per our third research question *RQ3*. For instance, in the case of the DApp *Simpli Finance*, the component becomes disconnected when the functions *safeDecreaseAllowance* and *withdraw* are removed, respectively in charge of controlling the amount of tokens that can be withdrawn from an account and actually withdrawing them. This critical threshold consistently appears to be around 2% indicating that DApps are susceptible to potential targeted attacks. The results show a similar pattern when nodes are removed based on their degree centrality instead of betweenness centrality, indicating that the network's response to targeted removal is consistent across different centrality measures. This result is also confirmed by the power-law degree distribution of nodes within the largest components, in Fig. 24 in Appendix D. In a small world network with a degree distribution following a power-law, deletion of a random node rarely causes a dramatic increase in the average path length, because most shortest paths between nodes flow through supernodes, and if a peripheral

[2] The 2% threshold is not imposed, but emerges from the experiment by monitoring when the largest connected component in the functions network becomes disconnected, upon removal of functions.

node is deleted it is unlikely to interfere with passage between other peripheral nodes. As the fraction of peripheral nodes in a small world network is much higher than the fraction of supernodes, the probability of deleting an important node is very low.

5 Discussion & conclusion

5.1 Mapping network analyses to security issues in DApps

By performing targeted network analyses, such as centrality measures and resilience analysis, we have provided new insights into the structural weaknesses of DApps and their implications for security. One of the key findings of our study is the importance of network resilience in ensuring the overall security of DApps. Our resilience analysis reveals that DApps are highly vulnerable to targeted attacks on functions with high betweenness centrality. These functions act as critical bridges between different components of the DApp, and their failure can lead to a significant disruption of the DApp's functionality. For instance, in the case of the DAO hack, the attacker exploited a reentrancy vulnerability in a function with high betweenness centrality to drain funds from the contract repeatedly. This example highlights the need for DApp developers to prioritize the security auditing of such critical functions and implement robust measures to prevent reentrancy attacks.

Another important aspect of our network analysis is the identification of functions with high clustering coefficients. These functions are tightly interconnected within their local communities, and this may indicate a lack of proper access control mechanisms. In the context of DApps, inadequate access control can allow attackers to manipulate sensitive data or perform unauthorized actions. For example, in the case of the Parity Wallet hack [28], a vulnerability in the access control mechanism of a library contract allowed the attacker to take control of multiple wallets and steal funds. This incident shows the importance of ensuring proper function segregation and implementing strict access control policies in DApps.

The community structure identified in our analysis has implications for the propagation of vulnerabilities. Tightly knit communities of functions may contain vulnerabilities that can quickly spread within the community, but remain isolated from the rest of the DApp. Conversely, functions that bridge communities (with high betweenness centrality) could potentially spread vulnerabilities across different parts of the DApp if compromised.

Our findings also have significant implications for DApp auditors and users. For auditors, our network-based approach can complement traditional code analysis techniques by providing a holistic view of the DApp's security. By analyzing the function interaction network, auditors can identify vulnerabilities that may be overlooked when examining individual contracts in isolation.

This can help auditors prioritize their efforts and focus on the most critical components of the DApp.

The small-world properties observed in DApp function networks suggest that information – including potential exploit techniques – can spread rapidly through the network. While this can be beneficial for efficient operation, it also means that once a vulnerability is exploited, its effects could quickly cascade through the DApp. This underscores the importance of robust security measures, particularly for functions that act as hubs in the network.

For users, our findings can serve as a basis for assessing the overall security and resilience of a DApp before deciding to interact with it. By considering the network properties of the

DApp, such as the presence of critical functions or tightly connected communities, users can make informed decisions about the potential risks associated with using the DApp itself.

Our analysis of function call frequencies and network centrality can also provide insights into gas optimization and potential Denial of Service (DoS) vulnerabilities. Functions with high betweenness that are frequently called may be prime targets for gas optimization to reduce overall transaction costs. However, these same functions, if not properly optimized or protected, could be exploited in DoS attacks that aim to consume excessive gas and disrupt the DApp's functionality.

By analyzing the evolution of DApp function networks over time (through different versions or updates), we can identify potential security risks associated with code changes. Functions that rapidly gain centrality in the network may require additional scrutiny, as they represent new or expanded attack surfaces. Conversely, functions that lose connections over time but remain in the codebase might represent deprecated functionality that could present overlooked vulnerabilities. Our network analysis results provide insights into the architectural vulnerabilities of DApps and their potential impact on security.

Based on our findings, we recommend that DApp developers and auditors pay special attention to functions with high betweenness centrality, as these represent critical points of failure. Implementing additional security measures such as multi-signature requirements or time-locks for these functions could significantly enhance the overall security of the DApp. Furthermore, regular network analysis of the DApp's function interactions can serve as an early warning system for potential architectural vulnerabilities introduced during development or updates.

5.2 Conclusion

We considered decentralised applications (DApps) of varying sizes, with different purposes, and deployed on various blockchain platforms, discovering consistent structural characteristics of contract and function networks across all of them. This consistency suggests that different development teams in the blockchain community adopt similar coding practices for smart contract design and development, regardless of the specific blockchain in use. In order to assess the resilience and security of DApps, we analysed the relationship within functions ad contracts. Ensuring that each function has a precise role when interfacing with a contract mitigates the effects of faults, but also facilitates a more systematic traceability, and verification of interactions for identifying anomalous behaviors. If modifications are required, all interactions of a given function can be redirected to an alternative contract, thus preserving the system's functionalities. In addition, by analysing and optimizing the frequency and nature of interactions between functions and their respective contracts, one can potentially minimize operational expenses like gas fees, consequently increasing the overall efficiency of the DApp. The analysis of both contract and function networks within decentralized applications reveals interesting structural insights and interaction dynamics.

The networks of contracts interactions exhibit high sparsity, and a significant portion of the links consists of self-loops, suggesting that contracts within a DApp primarily interact with themselves. DApps are built with a focus on modular, self-sufficient contracts, forming distinct communities with limited external interactions and prioritizing security, fault isolation, and functional boundaries within DApps.

We analyse the networks of functions interactions as well, in order to inspect code interactions within the DApp at a finer resolution. Contracts tend to distribute responsibilities among multiple functions, a practice that can be seen as a proactive measure against potential issues when relying solely on a single function to perform a complex task. Also the function networks maintain consistent characteristics that transcend the DApp's intended purpose. In this case there is a greater level of interconnection and a more intricate web of interactions among functions within the same DApp. DApps function networks exhibit a core largest component comprising functions interacting across multiple contracts, and encoding the core DApps' functionalities. Across all DApps, the organisation in distinct sub-components emerges, representing groups of functions with different - but secondary - tasks, defined within the same contract. Looking at the emergent structural organisation into core and secondary components, one may speculate that there is a coordinated and planned development of the core components, while the secondary contracts (interacting only with themselves) are disconnected parts added on an as-needed basis. An analysis of the timeline of development, monitoring code changes on Github, could shed further light on the architectural design and growth of the DApp infrastructure.

The largest component, containing interacting functions defined in different contracts, encodes the core functionality of the DApp. The core functions exhibit a high clustering coefficient and a low average shortest path length, resembling a small world model and suggesting significant local interactions, efficient information flow, and connectivity between functions. In the context of DApps, the small world structure implies that even when functions are not directly connected, efficient pathways for communication and interaction exist. Nodes with a high degree, which are closely correlated with nodes exhibiting high betweenness centrality (indicating their role as intermediaries between communities), as revealed by the degree distribution analysis, are relatively few: this means that a hacking attack on a random function does not significantly impact the overall DApp's functioning. However, if functions with high betweenness centrality are targeted in an attack, the largest component would immediately become disconnected and the information would stop flowing. This critical threshold consistently appears to be the 2% of the total number of nodes. These findings emphasize that DApps are susceptible to potential targeted attacks, pointing to the importance of implementing robust strategies to mitigate potential vulnerabilities in these specific functions and guarantee their continued functionality.

Identifying the core areas and bridges within these networks allows us to monitor critical sections of the DApp, anticipate potential vulnerabilities, and explore ways to optimize computational costs. These patterns in the data are found through the examination of network structures. The subsequent phase should include the analysis of actual transaction data, monitoring the execution of on-chain code, and the actual usage of functions and contracts by DApps' users.

Moreover, further information and metrics to assess the characteristics of functions can be overlaid onto the network information, such as those extracted and analysed [29, 41]. This will allow further analyses of the quality of interactions among function calls. To comprehend the underlying reasons for the emergence of these patterns, it is essential to expand the information regarding the network structures with additional metrics, such as complexity costs or the number of lines of code in functions. As an extension to the current analysis, it is also interesting to look more closely at the edge directionality in the functions' network, to identify particular structures within the network related to further potentially exploitable vulnerabilities.

Indeed, not all vulnerabilities that exist are actually exploited [42], but they still constitute a potential threat to the normal functioning of the platform. The exploitation of vulnerabilities in targeted attacks, leads to a decrease in trust in decentralised platforms, hindering users' adoption [43] and institutional investors' support [44].

Appendix A: List of DApps

The categorisation of DApps used in this work was done following the classification proposed by BitDegree and DappRadar. The dataset analysed comprises the set of dApps listed in Table 4.

Table 4 List of dApps grouped by blockchain and category.

Blockchain	Category	dApps
Ethereum	Collectibles	Async Art; Audius; Cryptovoxels
Ethereum	DeFi	1inch Network; Aave; BondAppetit; BT.Finance*; Compound; Etherisc; Naos Finance; Nexo.io; Nexus Mutual; Openleverage*; Origin Dollar; Polymarket; PWN; Rari Capital; Rocket Pool; SIGH Finance; Simpli Finance*; SPICE; SWAPP Protocol*; Synthetix*; TokenSets; Tsunami; UMA*
Ethereum	Exchanges	Balancer; Brickblock; Loopring Exchange; Plexus*; Popsicle V3 Optimizer*; ThorusFi*; Uniswap
Ethereum	Gambling	DSG; Stargate
Ethereum	Games	Axie Infinity; DARK FOREST; Gods Unchained; Marble.Cards
Ethereum	High-risk	Proof of Fair Launch
Ethereum	Marketplaces	Foundation; Fractional; OpenSea; Rarible; SuperRare
Ethereum	Other	Aragon Fundraising; AZTEC; Ethereum Name Service; Polymath; PoolTogether; Tornado Cash
Binance Smart Chain	DeFi	Atlantis; BabySwap; FarmHero; KillSwitch; Pandora; Rikkei Finance; Tranchess; Venus
Binance Smart Chain	Gambling	LuckyChip
Binance Smart Chain	Gaming	NomadLand
Astar	DeFi	ArthSwap
Polygon	DeFi	Angle
Polygon	Gambling	Reality Cards
Shiden	DeFi	Standard Protocol
ThunderCore	DeFi	Staking Pool

(*): cross-chains DApps
Popsicle V3 Optimizer: Ethereum, Avalanche, Fantom, Binance, Polygon;
BT.Finance: Ethereum, Binance Smart Chain;
Simpli Finance: multichain;
ThorusFi: multichain;
SWAPP Protocol: Ethereum, Binance Smart Chain;
Openleverage: Ethereum, Binance Smart Chain;
Plexus: Ethereum, Binance Smart Chain, Optimism, Polygon;
Synthetix: Ethereum, Optimism;
UMA: Ethereum, Optimism.

Appendix B: Contracts networks

In this section, we show further examples of DApp contracts networks. In Fig. 18, panel (A), we show *Balancer* Contracts Network, a DApp deployed in Ethereum and belonging

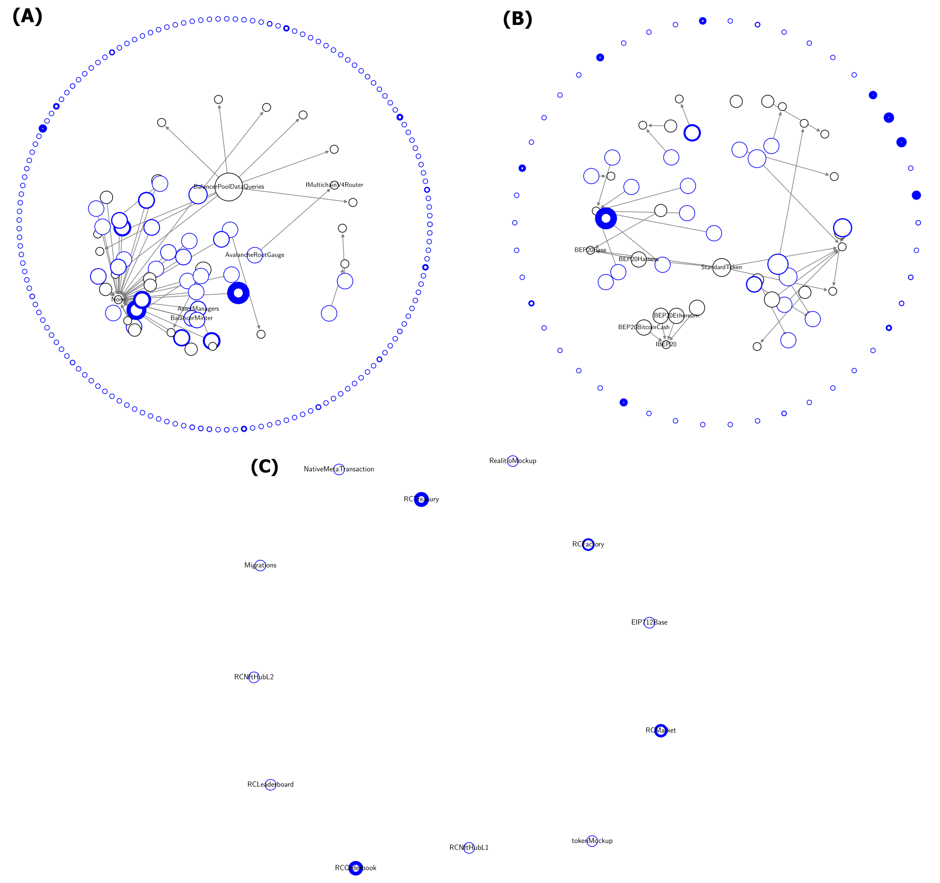
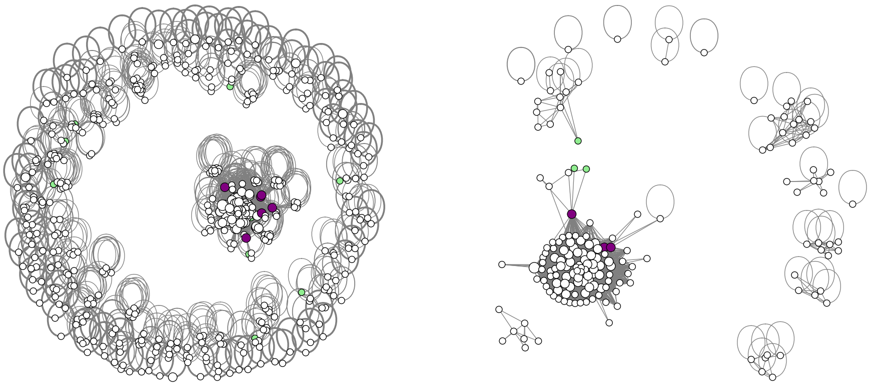


Figure 18 (A) *Balancer* Contracts Network (Ethereum - Exchanges). (B) *Venus* Contracts Network (BNB - DeFi). (C) *Reality Cards* Contracts Network (Polygon - Gambling). In blue, the nodes with self-loops. The width of the border of each blue node corresponds to the weight of its self-loop link. The names of the contracts with the highest betweenness centrality are listed

to the category exchanges. It is a large DApp consisting of 193 contracts. The total balance is \$223.7T and it is ranked #33 in the category Exchanges, and #359 in the General category in DApp radar. In blue, we highlight the nodes with self-loops, where the width of the border of each blue node corresponds to the weight of its self-loop link. The names of the contracts with the highest betweenness centrality are listed. In Fig. 18, panel (B), we show *Venus* Contracts Network, a DApp deployed on Binance (BNB) and belonging to the DeFi category. It is classified as a large DApp with 83 contracts and a total balance of \$574.32B. It is ranked #5 in DeFi and #10 in General. In Panel (C), we present *Reality Cards* Contracts Network, deployed on the Polygon blockchain network and listed in the Gambling category. It is a rather small DApp with 12 contracts and a balance of $\$13,32k$. It is ranked #1221 in the Gambling category.

Appendix C: Functions networks

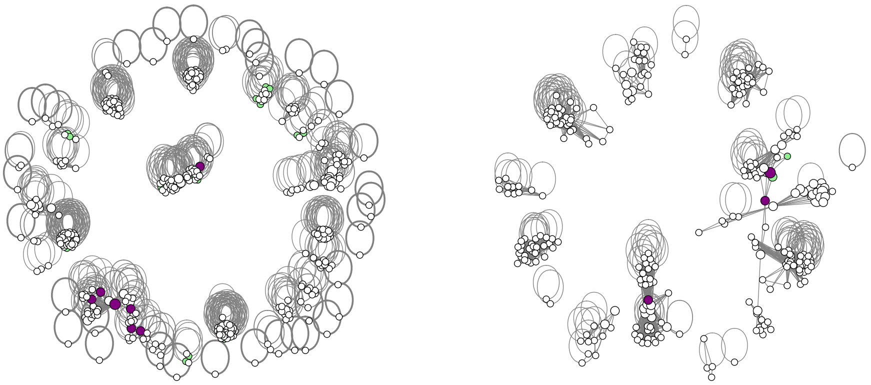
In this section, we show further examples of DApp function networks before and after applying the filter. We consider as in Sect. B the following DApps: *Balancer* (Fig. 19), *Venus* (Fig. 20), and *Reality Cards* (Fig. 21). We also highlight key functions and their role in the DApp within the networks.



(a) Network for functions' interactions pre-filter

(b) Network for functions' interactions post-filter

Figure 19 *Balancer* Functions Network (Ethereum - Exchanges). In purple the nodes with highest betweenness (i.e. *_addGauges*, *removeToken*, *removeAllowedAddress*), in green the nodes with highest clustering coefficient (i.e. *_returnLeftoverEthIfAny*, *addGaugesWithVerifiedType*, *_onSwapMinimal*). The purple functions are in charge of claiming the measure of liquidity provided by users or removing an address that was previously authorised for given transactions from the pool. The green functions return the ETHs that advance as a result of a given transaction, retrieve the status of the pool and verify that it is active



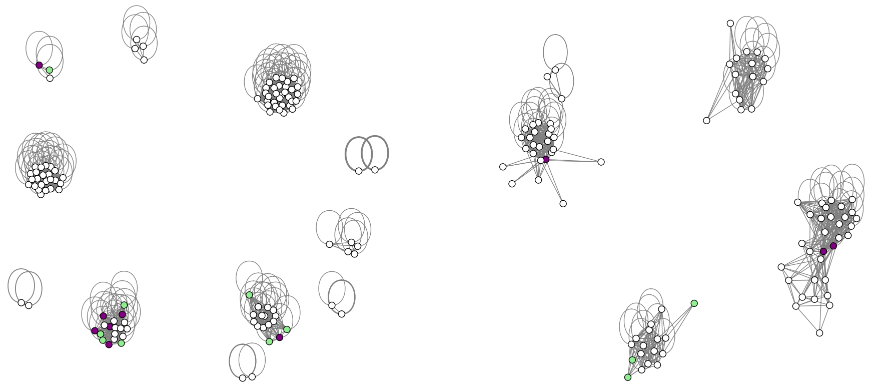
(a) Network for functions' interactions pre-filter

(b) Network for functions' interactions post-filter

Figure 20 *Venus* Functions Network (BNB - DeFi). In purple the nodes with highest betweenness (i.e. *reclaimToken*, *transferFrom*, *approve*), in green the nodes with highest clustering coefficient (i.e. *_burn*, *allocateTo*, *_transferOwnership*). The purple functions are in charge of transferring or claiming an amount of tokens from one address to another and confirming that it has been done successfully. The green functions destroy an amount of tokens relating to an account or transfer their ownership

Appendix D: Further results

In this section, we summarize the main structural traits of the principal DApps networks. We present in Table 5 the network analysis results for the highest-ranked DApps, selecting one example for each underlying blockchain protocol. We use as proxy of ranking the Unique Active Wallets (UAW) metric. UAW measures the number of individual Web3 wallets that have connected to a DApp over a given period of time. This metric is sometimes mistaken for a user count, however it differs from a regular user count because one



(a) Network for functions' interactions pre-filter

(b) Network for functions' interactions post-filter

Figure 21 *Reality Cards* Functions Network (Polygon - Gambling). In purple the nodes with highest betweenness (i.e. *addMarket*, *decreaseBidRate*, *changeApprovedAffiliatesOnly*), in green the nodes with highest clustering coefficient (i.e. *_postQuestionToOracle*, *claimCard*, *exit*). The purple functions are in charge of removing the bet based on its withdrawal and changing the status of a market affiliate. The green functions require a card during the game and end the current game

Table 5 Summary of main properties of the highest ranked DApps for each blockchain. The ranking is based on the Unique Active Wallets (UAW) metric.

DApp and Ranking	No. of Contracts	No. of Functions pre-filter	Functions Network Density	Functions Network Clustering Coefficient	No. of Components	Size of Largest Component	Diameter of Largest Component	
Ethereum	*Aave* - #2 in DeFi	107	392	0.28	0.69	11	104	4
Binance Smart Chain	*Venus* - #13 in DeFi	83	414	0.053	0.56	18	42	4
Astar	*ArthSwap* - #2844 in DeFi	14	63	0.33	0.38	1	58	4
Polygon	*Reality Cards* - #1214 in Gambling	12	110	0.16	0.68	5	32	2
Shiden	*Standard Protocol* - #2655 in DeFi	66	261	0.091	0.61	12	128	5
ThunderCore	*Staking Pool* - #517 in DeFi	45	224	0.074	0.53	11	46	3

person can have multiple wallets they use to connect with a DApp. The data regarding the ranking are available on DappRadar, already used for the classification of DApps in categories.

In Figs. 22 and 23 we include further analysis on cluster coefficients and diameter of the largest connected component for non-DeFi related DApps. Patterns similar to those observed by restricted the set to DeFi only DApps are present (see Fig. 12).

In Fig. 24 we show the degree distribution computed across all nodes in all DApps. The power-law degree distribution suggests that most nodes have few neighbors, while some supernodes (closely correlated with the nodes with highest betweenness) have a higher number of neighbors.

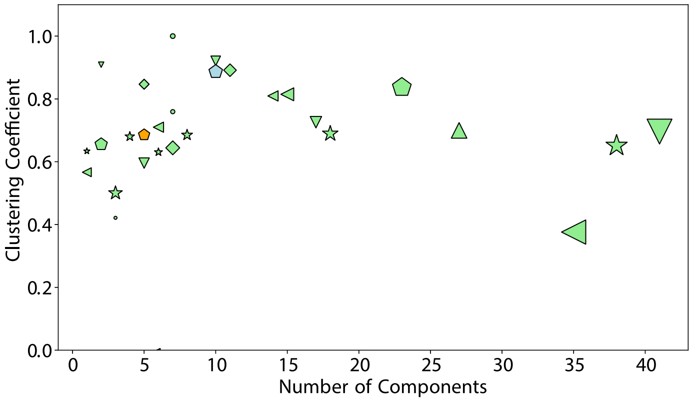


Figure 22 Scatter plot of the global clustering coefficient in the largest components vs the number of components. Each dot represents a DApp, and the size of the dot is proportional to the number of functions in the specific DApp. Collectibles (●); Exchanges (★); Gambling: (⬟); Games (⧫); High-risk (▲); Marketplaces: (▼); Other: (◀)

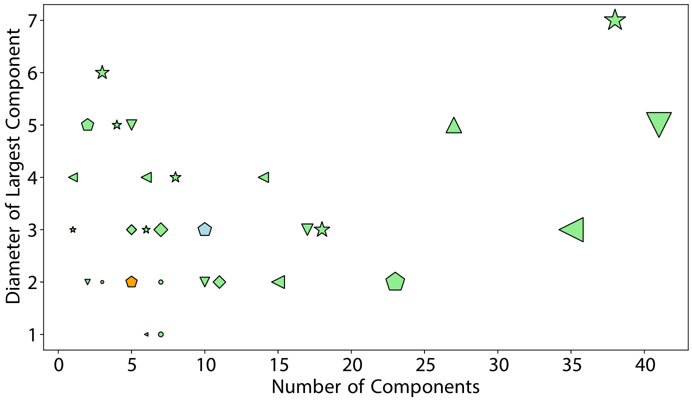


Figure 23 Scatter plot of the diameter of the largest connected component vs the number of components, for non-DeFi related DApps. Each symbol represents a DApp, and its size is proportional to the number of functions in the specific DApp. Collectibles (●); Exchanges (★); Gambling: (⬟); Games (⧫); High-risk (▲); Marketplaces: (▼); Other: (◀)

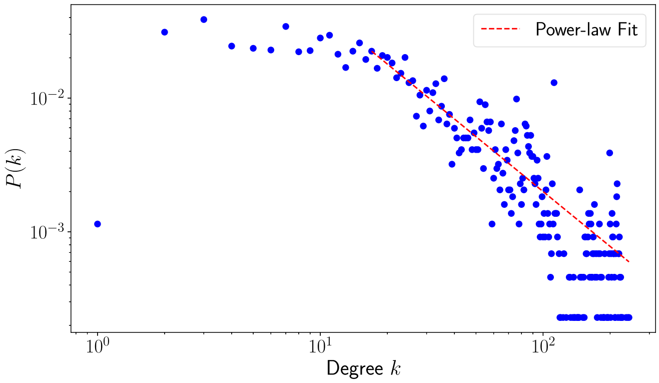


Figure 24 In blue, the degree distribution of the largest components is displayed. In red, a power-law fit of the distribution is shown, with $\alpha = 1.3$ and $x_{min} = 17$. The fit is performed using the Python library *powerlaw*, which implements the statistical method developed by Clauset et al. [45]

Abbreviations

DApps, decentralized applications; AST, Abstract Syntax Tree; DeFi, Decentralized Finance; TVL, Total Value Locked.

Acknowledgements

The authors gratefully acknowledge useful discussion and feedback on the manuscript from D. Vidal-Tomás, F. Caccioli and T. Aste.

Author contributions

All authors contributed to the study conception and design. S.A. and G.A. conducted the data collection. S.A. performed the data analysis. S. A. and S. B. wrote the manuscript. All authors discussed and interpreted the results, read and approved the final manuscript.

Funding

S.B., G.D., R.N. and M.O. acknowledge support from the Ethereum foundation grant FY23-1048.

Data availability

The datasets analysed during the current study can be retrieved using the tool presented in [30]. Smart contracts data of Dapps are publicly available from their respective Github repositories. To support future work in this area, we have made our dataset publicly available via the Zenodo repository https://zenodo.org/records/12731531 [10] and https://zenodo.org/records/13772792 [11].

Declarations

Competing interests

The authors declare that they have no competing interests.

Author details

[1]Dept. of Computer Science, University College London, London, UK. [2]Dept. of Business and Economic Sciences, University of Cagliari, Cagliari, Italy. [3]Dept. of Computer Science, Brunel University, London, UK.

Received: 19 February 2024 Accepted: 2 September 2024 Published online: 27 September 2024

References

1. Antonopoulos AM, Wood G (2018) Mastering Ethereum: building smart contracts and DApps. O'Reilly Media
2. Qian P, Liu Z, He Q, Huang B, Tian D, Wang X (2022) Smart contract vulnerability detection technique: a survey. Preprint. Available at arXiv:2209.05872
3. Oosthoek K (2021) Flash crash for cash: Cyber threats in decentralized finance. Preprint. Available at arXiv:2106.10740
4. Gudgeon L, Perez D, Harz D, Livshits B, Gervais A (2020) The decentralized financial crisis. In: 2020 crypto valley conference on blockchain technology (CVCBT). IEEE, pp 1–15
5. Wu K, Ma Y, Huang G, Liu X (2021) A first look at blockchain-based decentralized applications. Softw Pract Exp 51(10):2033–2050
6. He D, Deng Z, Zhang Y, Chan S, Cheng Y, Guizani N (2020) Smart contract vulnerability analysis and security audit. IEEE Netw 34(5):276–282
7. Zhao X, Chen Z, Chen X, Wang Y, Tang C (2017) The dao attack paradoxes in propositional logic. In: 2017 4th international conference on systems and informatics (ICSAI). IEEE, pp 1743–1746
8. Huang Y, Bian Y, Li R, Zhao JL, Shi P (2019) Smart contract security: a software lifecycle perspective. IEEE Access 7:150184–150202
9. Financial Stability Board. The financial stability risks of decentralised finance (2023) Accessed 29/12/2023 at https://www.fsb.org/2023/02/the-financial-stability-risks-of-decentralised-finance/
10. Ibba G, Aufiero S, Bartolucci S, Neykova R, Ortu M, Tonelli R, Destefanis G Decentralized Applications Network Structure [Data set]. Available at Zenodo repository https://doi.org/10.5281/zenodo.12731531
11. Aufiero S, Ibba G, Bartolucci S, Destefanis G, Neykova R, Ortu M Adjacency matrices for dApps contracts and functions network [Data set]. Available at Zenodo repository https://zenodo.org/records/13772792
12. Farmer JD, Gallegati M, Hommes C, Kirman A, Ormerod P, Cincotti S, Sanchez A, Helbing D (2012) A complex systems approach to constructing better models for managing financial markets and the economy. Eur Phys J Spec Top 214:295–324
13. Linkov I, Kott A (2019) Fundamental concepts of cyber resilience: Introduction and overview. Cyber resilience of systems and networks, 1–25
14. Soloviev VN, Belinskiy A (2019) Complex systems theory and crashes of cryptocurrency market. In: Information and communication technologies in education, research, and industrial applications: 14th international conference, ICTERI 2018, Kyiv, Ukraine, May 14-17, 2018. Revised Selected Papers 14. Springer, Berlin, pp 276–297
15. Lucchini L, Alessandretti L, Lepri B, Gallo A, Baronchelli A (2020) From code to market: network of developers and correlated returns of cryptocurrencies. Sci Adv 6(51):eabd2204
16. Bartolucci S, Destefanis G, Ortu M, Uras N, Marchesi M, Tonelli R (2020) The butterfly "affect": impact of development practices on cryptocurrency prices. EPJ Data Sci 9(1):21
17. Ferretti S, D'Angelo G (2020) On the Ethereum blockchain structure: a complex networks theory perspective. Concurr Comput, Pract Exp 32(12):e5493
18. La Morgia M, Mei A, Mongardini AM, Nemmi EN (2023) A game of nfts: characterizing nft wash trading in the Ethereum blockchain. In: 2023 IEEE 43rd international conference on distributed computing systems (ICDCS). IEEE, pp 13–24

19. Bovet A, Campajola C, Mottes F, Restocchi V, Vallarano N, Squartini T, Tessone CJ (2023) The evolving liaisons between the transaction networks of bitcoin and its price dynamics. In: Proceedings of blockchain Kaigi 2022 (BCK22), p 011002
20. Louridas P, Spinellis D, Vlachos V (2008) Power laws in software. ACM Trans Softw Eng Methodol 18(1):1–26
21. Potanin A, Noble J, Frean M, Biddle R (2005) Scale-free geometry in oo programs. Commun ACM 48(5):99–103
22. Kleinberg J (2000) The small-world phenomenon: an algorithmic perspective. In: Proceedings of the thirty-second annual ACM symposium on theory of computing, pp 163–170
23. Valverde S, Solé RV (2003) Hierarchical small worlds in software architecture. Preprint. Available at arXiv:cond-mat/0307278
24. Theodore C, et al (2014) Forecasting Java software evolution trends employing network models. IEEE Trans Softw Eng 41(6):582–602
25. Myers CR (2003) Software systems as complex networks: structure, function, and evolvability of software collaboration graphs. Phys Rev E 68(4):046116
26. Ortu M, Destefanis G, Hall T, Bowes D (2023) Fault-insertion and fault-fixing behavioural patterns in apache software foundation projects. Inf Softw Technol 158:107187
27. Zou W, Lo D, Kochhar PS, Le Dinh X-B, Xia X, Feng Y, Chen Z, Xu B (2019) Smart contract development: challenges and opportunities. IEEE Trans Softw Eng 47(10):2084–2106
28. Destefanis G, Marchesi M, Ortu M, Tonelli R, Bracciali A, Hierons R (2018) Smart contracts vulnerabilities: a call for blockchain software engineering? In: 2018 international workshop on blockchain oriented software engineering (IWBOSE). IEEE, pp 19–25
29. Ibba G, Khullar S, Tesfai E, Neykova R, Aufiero S, Ortu M, Bartolucci S, Destefanis G (2023) A preliminary analysis of software metrics in decentralised applications. In: Fifth ACM international workshop on blockchain-enabled networked sensor systems
30. Ibba G, Aufiero S, Bartolucci S, Neykova R, Ortu M, Tonelli R, Destefanis G (2023) Mindthedapp: a toolchain for complex network-driven structural analysis of Ethereum-based decentralised applications. IEEE Access 12:28382–28394
31. Parr T (2013) The definitive ANTLR 4 reference. Raleigh, the Pragmatic Bookshelf, 1–326
32. Harvey CR, Ramachandran A, Santoro J (2021) DeFi and the future of finance. Wiley, New York
33. Kong H (2023) Institute of Blockchain and Financial Association. Crypto crime report. Accessed 29/12/2023 from https://hkibfa.io/wp-content/uploads/2023/02/Crypto_Crime_Report_2023.pdf
34. Wu K (2019) An empirical study of blockchain-based decentralized applications. Preprint. Available at arXiv:1902.04969
35. Arisholm E, Sjoberg DIK (2004) Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. IEEE Trans Softw Eng 30(8):521–534
36. Sayeed S, Marco-Gisbert H, Caira T (2020) Smart contract: attacks and protections. IEEE Access 8:24416–24427
37. Zhou T, Ren J, Medo M, Zhang Y-C (2007) Bipartite network projection and personal recommendation. Phys Rev E 76(4):046115
38. Ángeles Serrano M, Boguná M, Vespignani A (2009) Extracting the multiscale backbone of complex weighted networks. Proc Natl Acad Sci 106(16):6483–6488
39. Newman M (2018) Networks. Oxford University Press, London
40. Burt RS (2018) Structural holes. In: Social stratification. Routledge, London, pp 659–663
41. Ibba G, Aufiero S, Bartolucci S, Neykova R, Ortu M, Tonelli R, Destefanis G (2024) A curated solidity smart contracts repository of metrics and vulnerability. PROMISE 2024: proceedings of the 20th international conference on predictive models and data analytics in software engineering
42. Perez D, Livshits B (2019) Smart contract vulnerabilities: does anyone care? pp 1–15. Preprint. Available at arXiv:1902.06710
43. Auer R, Farag M, Lewrick U, Orazem L, Zoss M (2023) Banking in the shadow of Bitcoin? The institutional adoption of cryptocurrencies. Center for Economic Studies and ifo Institute. CESifo Working Paper, No. 10355
44. Mungo L, Bartolucci S, Alessandretti L (2023) Cryptocurrency co-investment network: token returns reflect investment patterns. EPJ Data Sci 13(1):11
45. Clauset A, Rohilla Shalizi C, Newman MEJ (2009) Power-law distributions in empirical data. SIAM Rev 51(4):661–703

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.