Received: 1 April 2019 | Accepted: 17 August 2019

DOI: 10.1002/spe.2751

SPECIAL ISSUE PAPER

WILEY

A first look at blockchain-based decentralized applications

Kaidong Wu | **Yun Ma** | **Gang Huang** | **Xuanzhe Liu**

Key Lab of High Confidence Software Technologies, Ministry of Education, Peking University, Beijing, China

Correspondence
Yun Ma, Key Lab of High Confidence Software Technologies, Ministry of Education, Peking University, Beijing 100871, China.
Email: mayun@pku.edu.cn

Funding information
National Key R&D Program of China, Grant/Award Number: 2018YFB1004800; National Natural Science Foundation of China, Grant/Award Number: 61725201; Beijing Municipal Science and Technology Project, Grant/Award Number: Z171100005117002

Summary

With the increasing popularity of blockchain technologies in recent years, blockchain-based decentralized applications (DApps for short in this paper) have been rapidly developed and widely adopted in many areas, being a hot topic in both academia and industry. Despite of the importance of DApps, we still have quite little understanding of DApps along with its ecosystem. To bridge the knowledge gap, this paper presents the first comprehensive empirical study of blockchain-based DApps to date, based on an extensive dataset of 995 Ethereum DApps and 29,846,075 transaction logs over them. We make a descriptive analysis of the popularity of DApps, summarize the patterns of how DApps use smart contracts to access the underlying blockchain, and explore the worth-addressing issues of deploying and operating DApps. Based on the findings, we propose some implications for DApp users to select proper DApps, for DApp developers to improve the efficiency of DApps, and for blockchain vendors to enhance the support of DApps.

KEYWORDS

decentralized applications, empirical study, Ethereum, smart contract

1 | INTRODUCTION

Since the invention of the cryptocurrency Bitcoin by Satoshi Nakamoto in 2008,[1] blockchain technologies have been rapidly developed and have drawn lots of attentions from both academia and industry. A blockchain is a decentralized, distributed, and public digital ledger that is used to record transactions across many nodes so that any involved record cannot be altered retroactively, without the alteration of all subsequent blocks. Due to its advantages of decentralization, immutability, security, and transparency, the blockchain has become one of the most promising infrastructural technologies for the next generation of Internet-based systems, such as public services, Internet of Things (IoT), reputation systems, and security services.[2]

Essentially, the blockchain is a kind of distributed systems, providing the computation capability for applications to run on multiple computation nodes. Since blockchains have no centralized control but are maintained according to decentralized consensus models, applications on blockchains actually belong to decentralized applications (DApps for short), a special type of software where the application execution is not controlled by a single entity. In history, DApps usually refer to applications that run on the Peer-to-Peer (P2P) network of computers rather than a single centralized computer. Many famous DApps have ever been developed and widely spread, such as BitTorrent[3] for file sharing, BitMessage[4] for instant messaging, and Popcorn Time[5] for video streaming.

Blockchains provide a general computation abstraction via the mechanism of smart contracts, making it easy to develop DApps for various application contexts. For example, the Ethereum blockchain[6] provides Turing-complete smart contracts for developers to implement general-purpose programs. In consequence, with the prosperity of blockchains, more and more blockchain-based DApps have emerged, being adopted to almost all areas. According to a recent report,

the value of the biggest blockchain-based DApp market, Ethereum DApp market, has reached billions of dollars as of January 2019.[7]

Despite of the popularity of blockchain-based DApps, there is no comprehensive understanding of such a rising ecosystem. The industrial reports on blockchain-based DApps focus on only basic usage statistics,[8,9] such as the number of daily active users and the amount of transactions. Academic research efforts are mainly devoted on the underlying blockchain system[10-12] as well as the mechanism of smart contracts.[13,14] Few studies investigate the characteristics and development practices of blockchain-based DApps.

To bridge the knowledge gap, in this paper, we conduct the first comprehensive empirical study on blockchain-based DApps. We choose Ethereum, the largest and the most popular platform for running blockchain-based DApps, as our target. We select 995 popular Ethereum DApps, which contain 5,158 smart contracts, and retrieve all the corresponding 29,846,075 transactions occurred in 2018. Based on the dataset, we try to answer the following three research questions:

- **RQ1: How is the popularity of DApps distributed?** We explore the popularity of DApps by the number of unique users, transactions, and transaction volumes, and compare categories of DApps. We also examine the change of popularity as time evolves. By answering this question, we can provide an overview of the DApp market for all stakeholders in the DApp ecosystem.
- **RQ2: Are there any common practices of developing DApps?** We investigate whether DApps are open source and how smart contracts are organized in a DApp. By answering this question, we can reveal the development practices of current DApps.
- **RQ3: How much is the cost of DApps when running on the blockchain?** Running DApps have to pay fee to blockchain miners for deploying and executing smart contracts. By answering this question, we can provide some recommendations for developers to reduce the cost of DApps.

Our key findings are summarized as follows:

- **Popularity of DApps** (Sections 4.1 & 4.2). The distributions of the number of users, transactions, and transaction volumes against the popularity of DApps typically follow the Pareto principle, ie, a few DApps have substantial popularity. DApps with financial features have large influence on the market.
- **Growth of DApps** (Section 4.3). The number of DApps was first rapidly grown for the categories including Exchanges, Gambling, and Finance. As more DApps are developed, the number of DApps from the high-risk category increases significantly, leading to potential security issues.
- **Open source of DApps** (Section 5.1). Currently, the open-source levels of DApps are a bit diverse and not satisfactory. Only 15.7% of DApps are fully open source where the code of both the DApp and related smart contracts are available. In contrast, 25.0% of DApps are completely closed source. In general, DApps whose smart contracts are open source, usually have more transactions than others, indicating that open source can have potentially significant impacts on the popularity of DApps.
- **Usage patterns of smart contracts** (Section 5.2). 75% of DApps consist of only one smart contract. For DApps with multiple smart contracts, there are three usage patterns of smart contracts: *leader-member* where smart contracts are invoked with each other through internal transactions, *equivalent* where there is no invocation between smart contracts, and *factory* where the child contracts are deployed by a factory contract.
- **Cost of deploying smart contracts** (Section 6.1). The average deployment cost per smart contract for single-contract DApps is less than that for multicontract DApps. The deployment cost per smart contract is correlated with the line of code (LoC) and the number of functions (NoF) where NoF has more influence on the deployment cost.
- **Cost of executing smart contracts** (Section 6.2). In the median case, only 50% of the prepaid fee for executing smart contracts is actually used, leading to half of the prepaid fee being locked in transactions until they are confirmed. Contract executions with internal transactions cost more than those without internal transactions.

To the best of our knowledge, our work is the first comprehensive analysis of blockchain-based DApps to date. Our findings present an overview of Ethereum DApp market, motivating future research and development. Specifically, the findings show strong implications for multiple stakeholders of the market. For example, end users can choose suitable DApps according to DApp distribution; end users and developers can set proper amount of prepaid fee for executing smart contracts to avoid their assets being locked during contract executions; DApp developers can understand DApp life cycle better and use the suitable pattern to design the DApp architecture and smart contracts; blockchain vendors can optimize some mechanisms to serve DApps better.

The remaining of this paper is organized as follows. We introduce the background of the Ethereum blockchain and Ethereum DApps in Section 2. We describe our dataset in Section 3. We characterize the DApp popularity in Section 4, analyze features in the development of DApps in Section 5, and investigate the cost of deploying and executing DApps in Section 6. We summarize our findings and implications in Section 7. We survey related work in Section 8, and conclude the paper in Section 9.

2 | BACKGROUND

We choose the largest DApp market, Ethereum DApps, to conduct our empirical study. In this section, we give some background knowledge of the Ethereum blockchain and the Ethereum DApps.

2.1 | Ethereum blockchain

The blockchain is a ledger system based on a P2P network, keeping records of transactions that represent value transfers between accounts. In the network, all nodes receive transactions, pack them into a block that is linked to the previous block, and broadcast the block. According to the consensus mechanism, if most nodes receive and accept a block, then the block will be a part of the ledger.

Ethereum is the second-generation blockchain after Bitcoin. The ledger of Ethereum is used to support the cryptocurrency, Ether (ETH).

In Ethereum, there are two kinds of accounts: user accounts and smart-contract accounts. The user accounts represent participants, including callers (who call functions of smart contracts), deployers (who deploy smart contracts on Ethereum), and miners (whose nodes work to do contribution to the ledger). The smart-contract accounts represent the smart contract that is a type of programs that are saved in and able to run on blockchains, called chaincode (code on chain) as well.[15,16] Ethereum is the first blockchain providing Turing-complete programming language to develop smart contracts.

Transactions can be classified from two dimensions:

On the one hand, according to the *data in the transaction*, transactions can be divided into ETH transfers and contract executions. An ETH transfer represents that a user account transfers some ETH to another one. A contract execution represents that an account calls a function of a smart contract with some data as the input and some ETH as the fee for executing the contract.

On the other hand, according to the *transaction initiator*, transactions can be divided into external transactions, which are initiated by user accounts, and internal transactions, which are initiated by smart contract accounts.

Figure 1 shows the relationship between the two classifications. If the target account of a transaction is a user account, the transaction belongs to ETH transfer. If the target account of a transaction is a smart-contract account, the transaction belongs to contract execution.

Accounts have to pay fee for all transactions. These fees are uniformly called gas in the Ethereum ecosystem.

2.2 | Ethereum DApp

Ethereum blockchain provides computation and storage capabilities via the mechanism of smart contracts. Therefore, Ethereum DApps can deploy smart contracts to use the capabilities provided by Ethereum to implement business logics. In theory, all the processes and data of a blockchain-based DApp should be handled and stored on the blockchain for pure decentralization. However, due to the performance bottleneck of state-of-the-art blockchain systems, current DApps usually implement only parts of their functionality on the blockchain. As a result, three kinds of architectures are adopted by Ethereum DApps in practice as shown in Figure 2: direct, indirect, and mixed. For DApps of the direct architecture (Figure 2A), the client directly interacts with smart contracts deployed on the Ethereum. DApps of the indirect architecture (Figure 2B) have back-end services running on a centralized server, and the client interacts with smart contracts

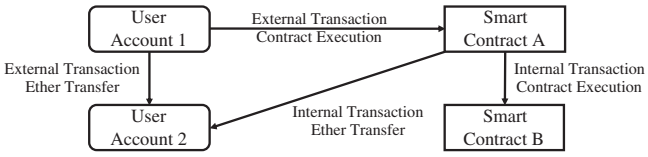

FIGURE 1 Relationship between the two classifications of transactions

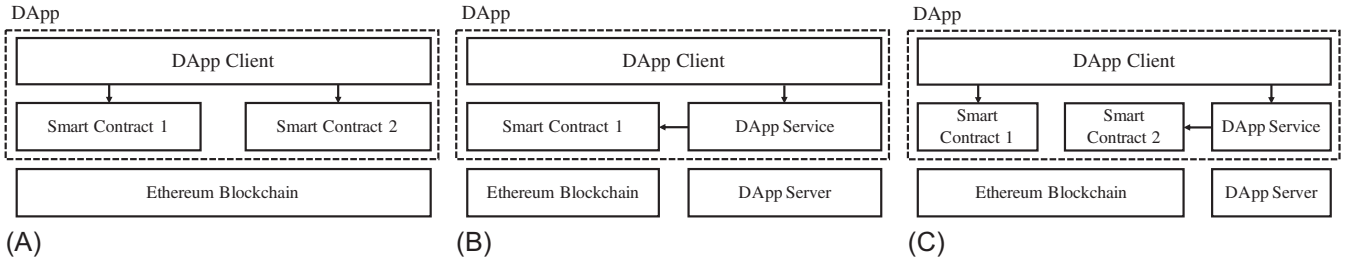

FIGURE 2 Three kinds of decentralized application (DApp) architectures. A, Direct; B, Indirect; C, Mixed

through the server. DApps of the mixed architecture combines the preceding two architectures where the client interacts with smart contracts both directly and indirectly through a back-end server.

Solidity is the programming language for developing smart contracts in the Ethereum community. It is a JavaScript-like language, in which there are contracts (such as classes), functions, and events. The source code of a smart contract is compiled into bytecode to be deployed on Ethereum. After deployment, the smart contract will get an address.

Smart contract deployment. All accounts can deploy smart contracts. Sometimes developers use a smart contract to deploy other child contracts. The usage pattern is called "factory pattern" and will be discussed in Section 5.2.

Cost of smart contracts. As mentioned above, accounts have to pay gas for every transaction. Cost of a smart contract consists of two parts: deployment and contract execution. The deployment of a smart contract can be seen as a contract execution of calling a special function *constructor()*. In this paper, we separate the deployment cost out of the general execution cost to study the two kinds of costs, respectively. The deployment cost can represent the complexity of the contract. Since executions are uniformly encoded in transactions, and finally packed into blocks, the total gas sent in transactions of a block is limited. Therefore, a contract execution that costs more gas than the limit will fail and change nothing.

3 | THE DATASET

In this section, we introduce the dataset that we collect to conduct our empirical study.

Since there is no official app marketplace for blockchain-based DApps such as App Store for iOS apps or Google Play for Android apps, we resort to third-party collections of DApps. Specifically, we choose DApps from *State of the DApps*,[17] which is a privately funded and independent website that collects DApps from major blockchain systems. Established in 2017, *State of the DApps* has grown to be the main directory of DApps where DApp developers can submit their DApps to get published. It is also worth to mention that *State of the DApps* is referenced by the official homepage of Ethereum. Thus, Ethereum DApps from this site are representative to understand the whole ecosystem.

We retrieve all the 1,749 Ethereum DApps that can be found on *State of the DApps* as of January 2019. For each DApp, we collect the following information.

- **Basic information of the DApp**. Including the name, the category it belongs to, and the first publishing time.
- **Smart contracts that the DApp consists of.** *State of the DApps* allows developers to specify the addresses of smart contracts that the published DApp consists of. Therefore, we can retrieve such information for some DApps. Then, we try to obtain the source code of each smart contract on *Etherscan*,[18] which is a block explorer and analytic platform of Ethereum where developers can submit the source code of their smart contracts.
- **Transactions related to the DApp.** We get blocks submitted in 2018 from Ethereum blockchain, and retrieve related contract executions in these blocks by checking addresses of participants. For each transaction, we extract addresses of senders and receivers, gas sent, input data, status (whether the contract execution succeeds), and how many ETHs transferred. Related internal transactions and gas used of external transactions are extracted by simulated executions, and double checked by being compared with results from Etherscan.

To deeply investigate how DApps utilize the capabilities provided by the blockchain, we filter those DApps that developers do not provide the addresses of smart contracts for, because we cannot retrieve smart contracts just by DApps' information and analyze how they work. After filtering, our dataset has 995 DApps left, which are used for the study in the following sections. Table 1 shows the statistics of our dataset.

DApps	995
Smart contracts	5,158
Open-source smart contracts	2,568
Transactions	29,846,075
User Accounts	2,199,059
Transaction volume (ETH)	9,057,344,360

Abbreviations: DApps, decentralized applications; ETH, Ether.

TABLE 1 Statistics of the dataset used in the empirical study

We get 5,158 smart contracts from lists that DApps provide, among which 2568 contracts' source code can be retrieved. There are 29,846,075 related transactions initiated in 2018, by which 2,199,059 user accounts transfer 9,057,344,360 ETHs.

4 | POPULARITY OF DAPPS

In this section, we try to describe an overview of the Ethereum DApp market and answer RQ1, ie, how is the popularity of DApps distributed? We first study the popularity by some metrics, and then, compare DApps in different categories. Finally, we examine the growth of the number of DApps as time evolves.

4.1 | DApp distribution

We use three metrics to measure the popularity of DApps: (1) the number of unique addresses (user accounts); (2) the number of transactions; (3) the amount of transaction volume (in ETH).

4.1.1 | Popularity by users

We extract all the user accounts occurred in the transactions of a DApp as the number of users that the DApp has.

Figure 3A shows the cumulative distribution function (CDF) of the percentage of DApp users against the DApp ranking by transactions. We can see that the DApp users follow the Pareto principle, ie, less than 20% of DApps have almost all users. We can conclude that the more a DApp is used, the more users it has.

We also explore the distribution of users of each DApp. Figure 3B indicates that about 80% of DApps are used by only less than 1000 users.

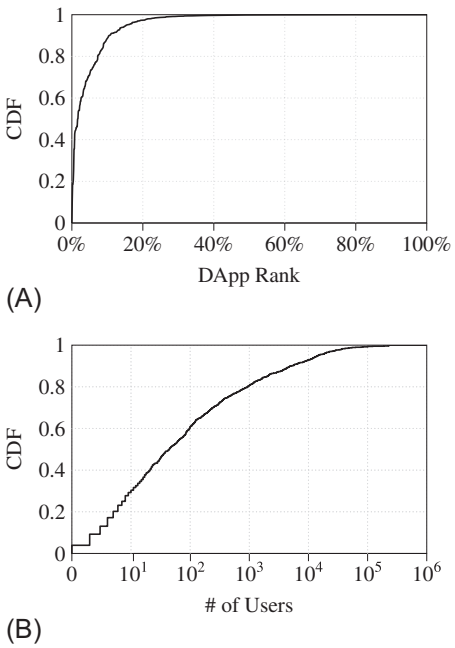

FIGURE 3 Decentralized application (DApp) popularity by users. A, Percentage of users against DApp rank; B, Users of a DApp. CDF, cumulative distribution function

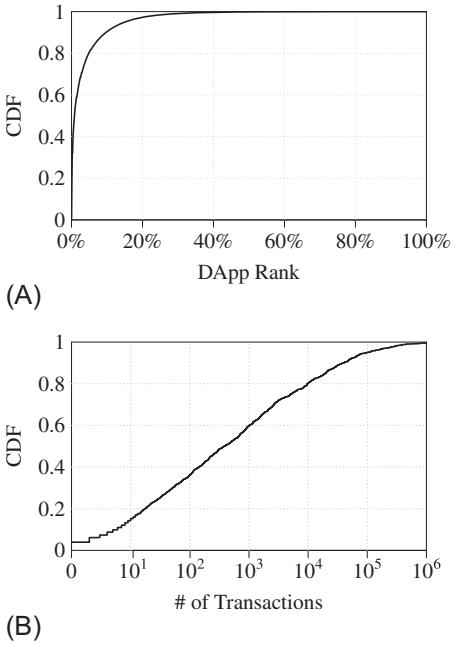

FIGURE 4 Decentralized application (DApp) popularity by transactions. A, Percentage of transactions against DApp rank; B, Transactions of a DApp. CDF, cumulative distribution function

4.1.2 | Popularity by transactions

Because all transactions are added to the block with paying gas, transactions can be seen as real behaviors of users. Figure 4A shows the CDF of the percentage of transactions of DApps against DApp ranking by transactions. We can find that a few (about 5%) DApps have 80% of transactions. It follows the Pareto principle as well.

Figure 4B shows that over 80% of DApps have less than 10,000 transactions. Such findings indicate a "long-tail" of DApps that are rarely active.

4.1.3 | Popularity by transaction volumes

In Ethereum, each transaction needs to pay gas, charged by ETHs, to miners, but they can transfer zero ETH. Therefore, DApps having many transactions may not be economic beneficial. Therefore, we study the actual transaction volume of DApps. Figure 5A illustrates the CDF of the percentage of DApps' transaction volumes against the DApp rank. We can find that the Pareto principle also holds for DApp transaction volumes.

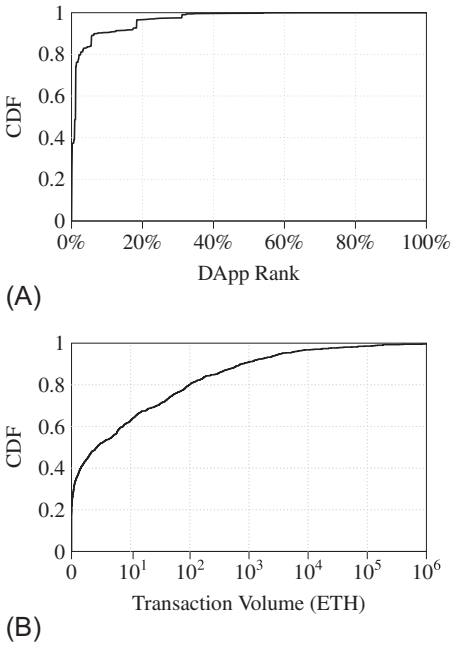

FIGURE 5 Decentralized applications (DApp) popularity by Ether (ETH) trade. A, Percentage of transaction volumes against DApp rank; B, Transaction volume of a DApp. CDF, cumulative distribution function

Figure 5B shows that about 60% of DApps have only less than 10 ETHs (about 6000 dollars in 2018) a year. 27.25% of DApps get few ETHs.

4.2 | DApp category

In *State of the DApps*, DApps are divided into 17 categories: Exchanges, Games, Finance, Gambling, Development, Storage, High-risk, Wallet, Governance, Property, Identity, Media, Social, Security, Energy, Insurance, and Health. The categorization is similar to that of mobile apps in Google Play[19] but simpler and rougher.

Some categories are not used to categorize mobile apps, such as Finance and Exchanges, which are two typical categories of DApps. Because of the financial feature of Ethereum, cryptocurrencies (category Finance), cryptocurrency management services (category Wallet), and cryptocurrency markets (category Exchanges) are hot. Recently, smart contracts are used in gambling (category Gambling), collectible card games, and gambling games (category Games).

DApps from category Property are marketplaces and services for other products, such as software. In other categories, DApps claim that they provide distributed or decentralized services to meet users' requirements.

DApps from category Development claim that they provide services to help to use distributed computing services of blockchain and develop smart contracts and DApps. Category Storage is used to mark DApps that provide decentralized storage services, Compared with similar mobile apps, decentralization and robustness is placed on them.

High-risk seems to be a tag rather than a category. According to the work of Chen et al,[20] there are over 400 Ponzi schemes running on Ethereum. Therefore, category High-risk marks DApps in which users may take high risk, such as investment traps and Ponzi Scheme games.

Table 2 shows the popularity of DApps among different categories. We can find that category Games has the most DApps (29.5%) but 8.4% of users (seventh), the second most (19.5%) transactions, and 2.5% of transaction volume (fifth). DApps of this category have become a hot topic in Ethereum DApp market. Category Exchanges and Finance have many users (35.4% and 23.5%) and high ratio of transaction volume (61.5% and 25.6%). These DApps have great influence. Category Identity, Media, Social, Security, Energy, Insurance, and Health have fewer users and transactions. They represent new designs of DApps but their practicability still needs verification.

4.3 | Growth over time

Figure 6A shows how the number of DApps grew over time before January 1, 2019. In our dataset, the first DApp was published at April 22, 2015. The number of DApps grew significantly from May 2017. After February 2018, more DApps were published every day.

TABLE 2 Decentralized application (DApp) popularity by categories

Category	DApps		Users		Transactions		Transaction Volume	
	#	%	#	%	#	%	#	%
Exchanges	71	7.1%	778,031	35.4%	13,708,713	45.9%	5,570,026.10	61.5%
Games	294	29.5%	184,730	8.4%	5,834,574	19.5%	226,506.11	2.5%
Finance	93	9.3%	517,525	23.5%	2,571,729	8.6%	2,321,199.88	25.6%
Gambling	154	15.5%	77,790	3.5%	1,781,856	6.0%	448,476.51	5.0%
Development	30	3.0%	269,821	12.3%	1,154,346	3.9%	20,525.36	0.2%
Storage	13	1.3%	249,544	11.3%	1,031,779	3.5%	11.29	0.0%
High-risk	130	13.1%	47,186	2.1%	965,131	3.2%	370,543.94	4.1%
Wallet	17	1.7%	193,790	8.8%	787,160	2.6%	2,020.50	0.0%
Governance	18	1.8%	188,201	8.6%	633,211	2.1%	132.29	0.0%
Property	24	2.4%	46,707	2.1%	485,341	1.6%	47,421.00	0.5%
Identity	12	1.2%	82,251	3.7%	425,860	1.4%	5,330.45	0.1%
Media	48	4.8%	127,315	5.8%	403,055	1.4%	1,144.43	0.0%
Social	72	7.2%	88,355	4.0%	381,534	1.3%	10,065.19	0.1%
Security	14	1.4%	40,684	1.9%	127,550	0.4%	17,211.19	0.2%
Energy	3	0.3%	21,312	1.0%	95,025	0.3%	22,127.17	0.2%
Insurance	1	0.1%	5,755	0.3%	19,575	0.1%	0.52	0.0%
Health	2	0.2%	4	0.0%	9	0.0%	0.00	0.0%
All DApps	995		2,199,059		29,846,075		9,057,344.36	

Categories are sorted by Transactions

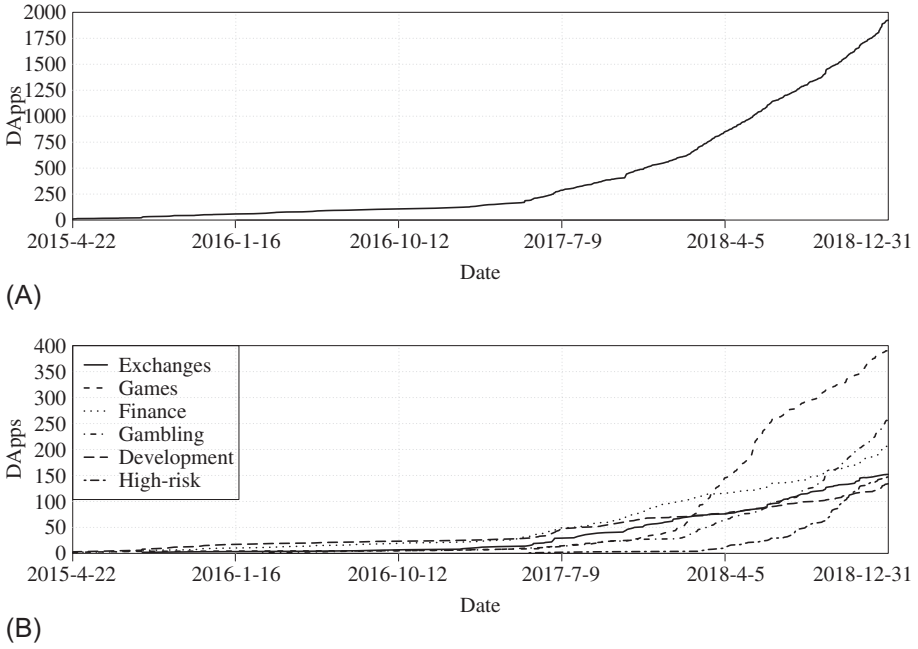

FIGURE 6 Decentralized applications' (DApps) growth over time. A, Growth of all DApps; B, Growth of DApps from category exchanges, games, finance, gambling, development, and high-risk

Figure 6B shows the growth of DApps of top six categories ranked by transactions. DApps from category Development and Finance were first published. At the beginning, all categories had similar trends. After May 16, 2017, the numbers of DApps from category Exchanges, Games, Gambling, and Finance grew rapidly. In February 2018, category Games had similar number with category Exchanges and Finance, then grew over the other two and led to the significant growth of the number of all DApps.

The first DApp from category High-risk appeared in May 2017. In February 2018, the number of DApps of High-risk grew more rapidly, of which the trend was similar with category Gambling's. At December 31, 2018, there were 154 Gambling DApps and 130 High-risk DApps in Ethereum DApp market.

Therefore, we can conclude that DApps from category Exchanges and Finance were hot early, and then, DApps with the financial feature such as gambling games grew rapidly. However, at the same time, high-risk DApps also became more and more.

5 | DEVELOPMENT OF DAPPS

In this section, we dig deeply into DApps and the corresponding smart contracts to answer RQ2, ie, are there any common practices of developing DApps? We first study whether DApps are open source, being able to be audited by any blockchain participant. Then, we investigate how developers implement smart contracts to support DApps.

5.1 | Open source

As illustrated in Figure 2, a DApp can be divided into two parts: on-chain part where smart contracts are implemented to use the capability of blockchains, and off-chain part where traditional programs are implemented to provide services to end users. Thus, we study two levels of open source of DApps: contract level (on-chain) and project level (off-chain).

Although all smart contracts can be directly retrieved from the blockchain, they are only in the form of bytecode, which is not readable. In practice, developers can submit the source code of smart contracts to block explorers or open-source code repositories. For the other parts of DApps, they can be open source through classical open-source channels, ie, GitHub.

According to whether DApps are open source at the contract level and the project level, there are six categories of the open-source degree of DApps, as shown in Table 3. Note that smart contracts may be partially open source, meaning that developers submit the source code of only part of smart contracts consisting of a DApp.

Figure 7 depicts the percentage of DApps in each open-source category. Only 15.7% of DApps are fully open source, and 25.0% of DApps are fully closed source. 54.9% of DApps do not make them project-level open source, meaning that they hide their business-related code. As for the contract-level open source, we can observe that there are 43.5% of DApps

TABLE 3 Categories of the open-source degree of decentralized applications (DApps)

	The DApp project is open source	The DApp project is not open source
Smart contracts are all open source	GA	NA
A part of smart contracts are open source	GP	NP
Smart contracts are all closed source	GC	NC

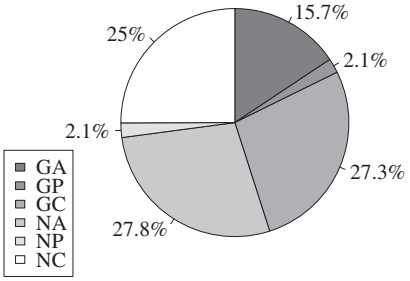

FIGURE 7 Percentage of open-source decentralized applications

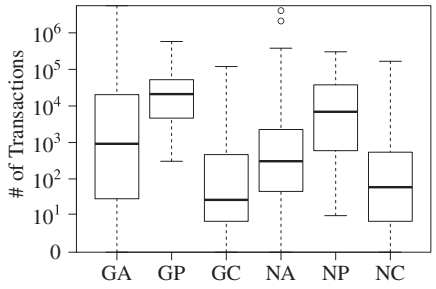

FIGURE 8 Distribution of number of transactions among different open-source categories

whose smart contracts are all open source. This result implies that developers tend to share the code of smart contracts rather than the DApps.

Then, we analyze the relationship between open source and popularity of DApps. Figure 8 depicts the distribution of the number of transactions among different open-source categories. We can find that DApps whose smart contracts are all closed source have smaller number of transactions. DApps whose smart contracts are all open source have higher maximum transactions. Therefore, we can conclude that the open source of smart contracts may improve the DApps' popularity.

5.2 | Usage patterns of smart contracts

Developers use smart contracts to keep data on the chain and do some operations. Sometimes the operations are too complex to be done by one smart contract, so that developers implement multiple smart contracts for a single DApp. Figure 9 shows the distribution of the number of smart contracts per DApp. We can find that over 75% (757) of DApps are supported by one smart contract. We denote these DApps as single-contract DApps.

About 25% of DApps are supported by multiple smart contracts (denoted as multicontract DApps), and almost all of them have less than 10 smart contracts. In the median case, a multicontract DApp has three smart contracts. The maximum number of smart contracts of a DApp is 1,881.

Considering features of smart contracts, we select two metrics to classify the usage patterns of smart contracts in multi-contract DApps: internal transactions among smart contracts and smart contract deployers. As shown in Table 4, we divide the smart contract usages into three patterns. Smart contract management in these patterns are illustrated in Figure 10. Internal transactions indicate that there is dependence among smart contracts. However, internal transactions initiated by smart contracts generated by smart-contract accounts are used to return ETH to their deployers where they have no dependence in functionality. Thus, we do not need to create a category to describe this usage pattern.

Leader-member pattern. In the leader-member pattern, a contract execution starts in a smart contract. Then, the entry smart contract (the leader) initiates internal transactions to other smart contracts (members). Therefore, in the leader-member pattern, contract executions usually have more deeper call stacks. For example, developers design smart contracts S, A, B, C, and set S as the entry contract. In a possible path, S is executed first, and then, it sends internal

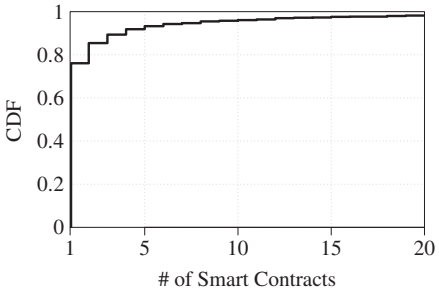

FIGURE 9 Distribution of the number of smart contracts per decentralized application. CDF, cumulative distribution function

TABLE 4 Classification of usage patterns of smart contracts

	Deployed by user accounts	Deployed by smart contract accounts
There are internal transactions	Leader-Member Pattern	NA
There are no internal transactions	Equivalent Pattern	Factory Pattern

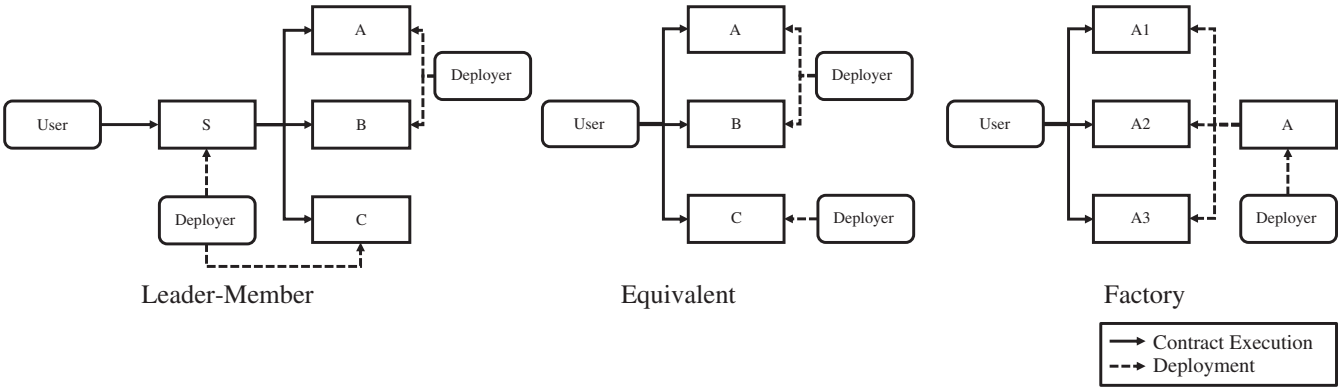

FIGURE 10 Usage patterns of smart contracts

transactions to A, B, C in order. Therefore, S is the leader, A, B, and C are members, and they form the leader-member pattern.

Equivalent pattern. The equivalent pattern is the simplest pattern. Developers design smart contracts separately, make them handle contract executions, and combine their functions in clients and/or servers. By only transactions on the blockchain, we cannot find dependence among them. Like smart contract A, B, C deployed by different deployers, they do not send internal transactions to each other, but they support the DApp together.

Factory pattern. In this pattern, a deployer deploys a smart contract and then make it deploy similar contracts to receive contract executions from user accounts. The process is like a factory producing products, so we call the pattern "factory pattern". The smart contract deployed first is the "factory contract", and smart contracts generated by the factory contract are called "child contracts". Figure 10 shows a case: deployer deploy smart contract A first, and make it deploy A1, A2, A3. Then, the three child contracts are able to be executed. Developers using the factory pattern can keep child contracts similar. This pattern is usually used in DApps from the category Gambling, whose developers use the factory contract to generate games with the same rules.

By checking internal transactions and deployers of smart contracts, we find leader-member pattern in 194 DApps and 199 smart contracts, equivalent pattern in 214 DApps and 1,539 smart contracts, and factory pattern in 28 DApps and 2,671 smart contracts. The leader-member pattern is not widely used. Developers are more likely to design independent smart contracts in functionality.

6 | COST OF SMART CONTRACTS IN DAPPS

The cost of smart contracts in DApps includes two parts: deployment cost and execution cost. Deployments and executions are done as transactions, which cost gas. Gas are paid with ETHs, and the amount of gas used is a measurement of the complexity of a contract execution. An account sends some gas in a contract execution, and then, gets the gas left when

the contract execution is confirmed. If the transaction has used all the gas sent from the initiator, the account receives an error information "out of gas" and lose all gas it sends.

To lower the costs of deployments and executions is important. In Ethereum blockchain, the total gas of a block is limited. A complex smart contract may cost too much gas so that it cannot be deployed, ie, the block will not contain the transaction. In addition, the higher the contract execution costs are, the lower the throughput of contract executions, and the longer users wait for confirmations of executions.

In this section, we investigate the gas that is actually used for deploying and executing smart contracts to answer RQ3, ie, how much is the cost of DApps when running on the blockchain? If not specified, in this section, the term "gas" represents the amount of gas that has been actually used in the contract execution.

6.1 | Deployment cost of smart contracts

As mentioned above, gas used in deploying a smart contract reflects the complexity of the contract. We compare the deployment costs among different usage patterns of smart contracts to study whether developers can leverage different patterns to lower the cost of deploying smart contracts. Then, we build a regression model to explore what factors influence the deployment cost.

6.1.1 | Deployment cost of different usage patterns of smart contracts

Figure 11 shows the distribution of smart contract deployment costs among different usage patterns. In general, smart contracts of the single pattern cost smaller than others, meaning that the smart contracts of single-contract DApps are simpler than those of multicontract DApps. The reason is that developers of single-contract DApps design limited functions in smart contracts, and may use back-end servers or relatively complex clients to implement the DApps.

Smart contracts of leader-member pattern and equivalent pattern have more deployment cost, meaning that contracts of these two patterns are more complex. The reason is that developers need smart contracts to do more operations. They split functions into multiple smart contracts and make them co-operate so that they could keep deployment cost of each smart contract at a low level.

Deployment costs of child contracts of the factory pattern are more concentrated around a value, because they are generated by some factory contracts. Child contracts generated by a certain factory contract have similar deployment costs.

6.1.2 | Influence factors

To explore what influences the deployment costs of smart contracts, we extract some features from smart contracts and their source code, and build a regression model to check relationships between features and the deployment cost.

From *Etherscan*, we can get Application Binary Interfaces (ABIs) and source code of smart contracts. The ABI describes a smart contract, including functions (normal functions and constructors) and events. The NoF and LoC could reflect the complexity of a smart contract and may influence the deployment cost.

First, each feature is separately used to build a linear regression model with deployment cost and do regression analysis by R.[21] The result is shown in Table 5. At the level of t<0.01, the deployment cost is obviously related to these features.

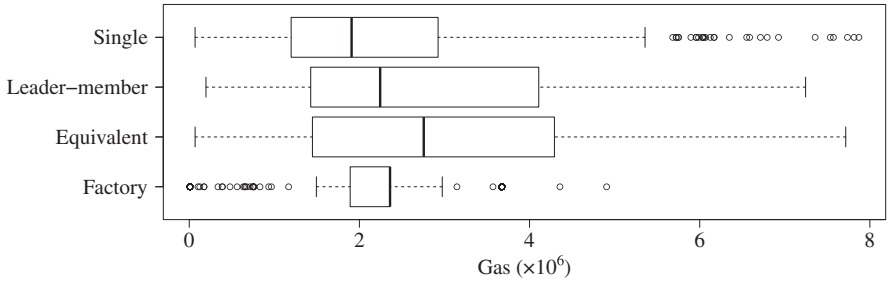

FIGURE 11 Deployment cost difference among smart contract usage patterns

TABLE 5 Result of two linear regression analyses

Feature	Estimate	Std. error	$Pr(>\|t\|)$	Multiple R-squared	P-value
NoF	53,389	1965	$< 2e-16$	0.2956	$< 2.2e-16$
LoC	1870	53.86	$< 2e-16$	0.4066	$< 2.2e-16$

Abbreviations: LoC, line of code; NoF, number of functions.

Then, we use all features with the deployment cost to do multiple regression analysis (Table 6). R^2 is significantly higher than results of two linear regression analyses above. The correlation coefficient of NoF is higher than LoC's, so we can conclude that NoF and LoC all influence the deployment cost, and the deployment cost is more related to NoF.

6.2 | Execution cost of smart contracts

In a contract execution, a user (the caller) sends some gas, input data and ETHs if they want, to call a function of the smart contract. Miners receive the transaction and add it into a block. When checking of the block is valid, miners will find the transaction and then execute it. They load the smart contract and its storage, and execute the function. They will load other smart contracts if the function calls other functions of other smart contracts, namely, create internal transactions. They count gas for any instructions. If all the gas is used or another error occurs, miners will stop the execution and mark the contract execution as "isError", meaning some errors in the execution and the contract execution fails. Unless all gas is used, gas left will be changed into ETHs and returned to the caller after the block is added into the chain, namely, the transaction is confirmed.

Therefore, to lower the costs of contract executions, callers should better send less gas in transactions. Meanwhile, developers should lower the complexity of contract executions. We check the gas sent and used in transactions, and explore if internal transactions and the usage pattern of smart contracts may influence the execution costs, to find a way to lower the cost.

6.2.1 | Gas sent in transactions

In practice, end users interact with clients, where some operations are translated to contract executions and sent to smart contracts. Contract executions (excluding the deployment) represent actual use of smart contracts. Users (sometimes the DApp maintainers) pay for them with ETHs. The gas left of a contract execution cannot be used until the transaction is confirmed. If users want to use their ETHs more flexibly, they should better send limited but enough gas.

Figure 12A shows the distribution of number of contract executions to gas left. We can see that 50% of contract executions have 100,000 gas left and 75% of contract executions have 200,000 gas left. About 5% of contract executions use all the gas. Such results indicate that over 100,000 gas are locked in about 50% of contract executions.

In Figure 12B, we can find that 70% of contract executions cost less than 100,000 gas, and 80% of contract executions cost less than 141,213 gas. Only about 5% of contract executions cost over 300,000 gas. Thus, if users want to request a contract execution, they can just send 141,213 gas to cover the cost in 80% of cases.

In the median case, users send 199,366 gas in contract executions, but only 99,366 gas is used and 100,000 gas left is locked in contract executions until transactions are confirmed.

6.2.2 | Contract executions with internal transactions

Internal transactions also cost gas. Because smart contracts cannot actively initiate internal transactions, they are triggered by external transactions. Internal transactions are included in external transactions that trigger them and the costs are

TABLE 6 Result of multiple regression analysis

Feature	Estimate	Std. Error	$Pr(>\|t\|)$
NoF	37,141.26	1,685.74	$< 2e-16$
LoC	1,514.68	50.34	$< 2e-16$

Multiple R-squared: 0.535, P-value: < 2.2e-16.
Abbreviations: LoC, line of code; NoF, number of functions.

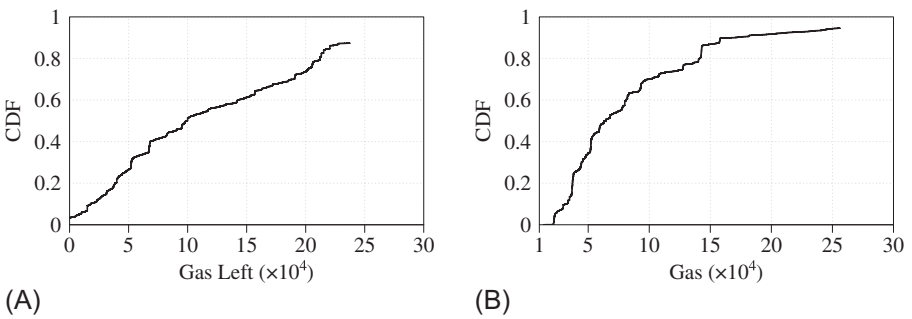

FIGURE 12 Gas of contract executions. A, Gas left of a contract execution; B, Gas used of a contract execution. CDF, cumulative distribution function

included in costs of contract executions as well. Thus, for similar contract executions, those that have internal transactions cost more gas than others.

Internal transactions are initiated for many reasons, but just a few of them are in design of DApps. Suppose there are two DApps, A and B, each of which is a multicontract DApp. A smart contract of A is triggered by a contract execution to initiate an internal transaction to B, and then, there are more internal transactions triggered among smart contracts of B. Internal transactions among smart contracts of B are not taken into account for developers of A, and they do not have to think about them. Therefore, we just consider internal transactions in DApp designs, which means internal transactions among smart contracts of DApps.

Figure 13 shows difference between the two types of contract executions. We can find that contract executions with internal transactions cost more gas than those without internal transactions.

6.2.3 | Difference among smart contract usage patterns

To compare execution costs of smart contracts among usage patterns, we have to use a metric to represent the average level of a smart contract's execution costs. We select an example from our dataset to show the distribution of a contract's contract executions and their costs.

IDEX[22] is a DApp categorized as Exchanges and has the most transactions in 2018. The contract selected is one of the two smart contracts it has. As shown in Figure 14, distributions of the contract's execution costs and numbers of invocations of functions are uneven. Neither arithmetical mean nor median can represent the average level.

For each function, gas used of its invocations is concentrated. Therefore, we take the number of invocations to functions into account, and define a new metric *agas* (average gas) to represent the average level of a smart contract's execution cost

$$agas = \frac{\sum mgas_i \cdot \text{count}_i}{\sum \text{count}_i}, \tag{1}$$

where $mgas_i$ represents gas median of function i, count_i represents the invocation count of function i.

Then, we compare the average gas of smart contracts of four patterns. According to Figure 15, in general, smart contracts of leader-member pattern and factory pattern cost more gas in contract executions. Smart contracts of equivalent pattern have a similar distribution of *agas* with those of single pattern.

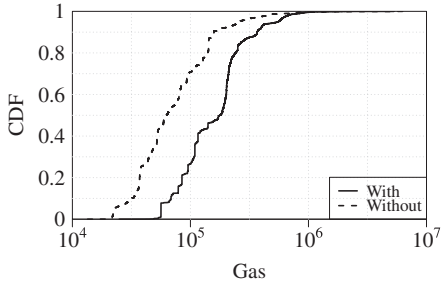

FIGURE 13 Execution cost difference between contract executions with and without internal transactions in decentralized applications. CDF, cumulative distribution function

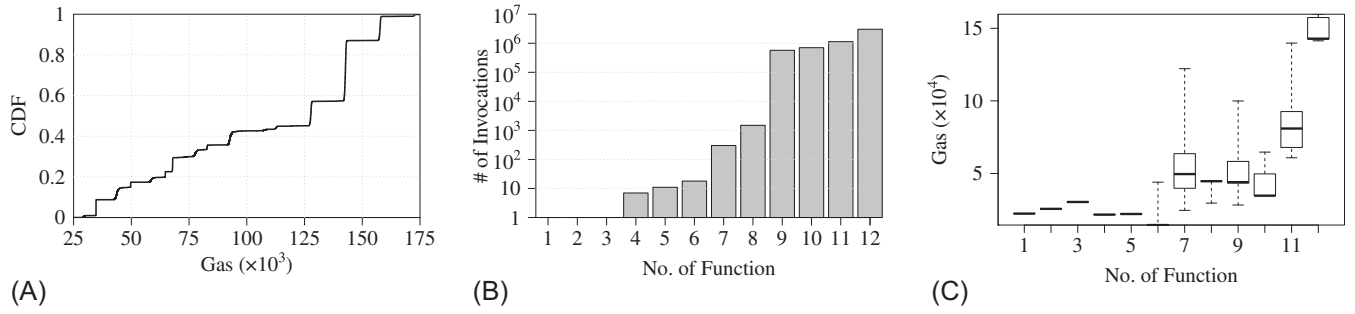

FIGURE 14 Contract 0x2a0c0DBEcC7E4D658f48E01e3fA353F44050c208 of IDEX. A, Gas of a contract execution; B, Distribution of numbers of invocations to functions; C, Distribution of gas used of each function. CDF, cumulative distribution function

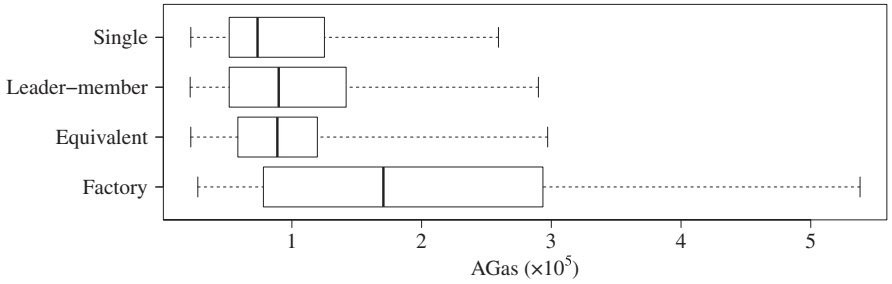

FIGURE 15 Execution cost difference among smart contract usage patterns

7 | FINDINGS AND IMPLICATIONS

Based on the results in preceding sections, we summarize the findings of our study and draw implications to the stakeholders in the ecosystem of blockchain-based DApps, including end users, DApp developers, and blockchain vendors. Table 7 shows all the findings and implications.

8 | RELATED WORK

To the best of our knowledge, our work is the first comprehensive study to understand the blockchain-based DApps. In this section, we first survey the related work of blockchain systems, and then introduce literature on P2P applications, which are the traditional type of DApps.

8.1 | Blockchain systems

The blockchain has become a hot research field. Researchers mainly focus on three research directions: the underlying mechanism, application, and data mining.

Underlying mechanism. Underlying mechanisms include consensus mechanisms and smart contract mechanisms. Many consensus mechanisms are proposed, such as PoW (Proof of Work),[1] PoS (Proof of Stake),[23] DPoS (Delegated PoS),[24] and Algorand[25] recently presented. Some classical consistency algorithms such as PBFT[26] (practical byzantine fault tolerance) are used as well. Furthermore, a few researchers make performance monitoring[10,11] and try to improve existing mechanisms.[27,28] Smart contract mechanisms attract experts of software engineering[29,30] and security. [13,14,31]

Blockchain application. For its decentralization, persistency, anonymity, and auditability, blockchain can be used in anonymous trading, persistence services and cross-organizational transactions. Therefore, blockchain technology has been widely used in finance service,[32] IoT,[33-35] information security,[36,37] edge computing,[38] and software engineering.[29]

Data mining. Thanks to the public accessibility and auditability of blockchain, researchers can do analysis based on transaction data for usage characteristics[39-41] and so on. The other accessible data are smart contracts, for example, their bytecode written in the blocks. There are some work analyzing these code to give advises for smart contract developers and blockchain users.[20,42,43] Researchers also try to decompile them into source code[44] so that more approaches to source code analysis can be used.

8.2 | P2P applications

Traditional DApps refer to applications on the P2P network,[45] on which there are lots of research efforts, including security, performance, and application.

Security. Security of P2P network and applications includes two parts: security of P2P network and possible harmful behaviors of users. For security of P2P network, researchers detect attacks,[46] use other technologies such as trusted computing[47] and design new protocols.[48] Some harmful behaviors in P2P applications are found as well.[49,50]

Performance. P2P applications are likely to become the burden of local network, especially P2P file sharing systems. Researchers monitor the performance[51,52] and try to improve these applications by optimizing application layer[53-55] and network layer.[56]

Application. P2P technology is widely used in many fields, such as instant messaging,[4] file sharing system,[57] development,[58] security,[59] and so on.

TABLE 7 Summary of findings and implications

Findings	Implications	Stakeholders
F1. On the blockchain such as Ethereum, DApps with financial features are more popular than others, such as DApps from the Exchanges, Finance, and Gambling categories. In contrast, other categories have relatively less users and transactions.	The development of blockchain-based DApp market is still at an early stage, mainly aiming at meeting the financial requirements DApps are not popular to satisfy more general application contexts.	End users
F2. Other categories can become popular, too, ie, Games. All DApps from these categories have financial features, such as in-app purchase services or collectible cards bought with Ethers.	Adding financial features, such as in-app purchase functions, may make DApps more popular.	DApp developers
F3. As more DApps are developed, the number of DApps from the high-risk category increases significantly.	Users have to be more careful to avoid Ponzi schemes or other similar investment frauds, even in DApps that seem not to be related to investment and gambling.	End users
F4. Only 15.7% of DApps are fully open source where the source code of the project and smart contracts is published. 52.3% of DApps' smart contracts are closed source.	DApps does not obey the blockchain's principles on opening and auditability. Blockchain vendors should better provide mechanisms to ensure the open source of DApps.	Blockchain vendors
F5. DApps whose smart contracts are open source have more transactions than those whose smart contracts are all closed source.	It is better for developers to make the smart contracts open source, which could improve the popularity of DApps.	DApp developers
F6. About 75% of DApps have only one smart contract. For multicontract DApps, most of the contracts are independent where only 199 (3.9%) smart contracts have internal transactions.	Developers should be careful in development. Because smart contracts cannot be modified after deployments, if some smart contracts break down, developers of multicontract DApps just need to replace the smart contract in which errors occur, but most developers have to redeploy all smart contracts of their DApps, namely the only one smart contract of single-contract DApps.	DApp developers
F7. In general, deploying a smart contract costs millions of gas. Both NoF (number of functions) and LoC (line of code) influence the deployment cost of smart contracts, and the deployment cost is more related to NoF.	Reducing NoF and LoC may lead to lower deployment cost of smart contracts. Reducing the number of functions could help more.	DApp developers
F8. 50% of contract executions have over 100,000 gas left. 80% of contract executions cost less than 141,213 gas.	To prevent more gas from being locked in contract executions, users can just send 141,213 gas. In 80% of the cases, such an amount of gas can cover the cost.	End users andDApp developers
F9. Contract executions with internal transactions in DApps generally cost more gas than those without internal transactions.	Splitting all functions into independent parts and using the equivalent pattern to organize smart contracts may reduce the execution cost. If functions are too complex to be split, it is better to keep just key functions in smart contracts and use a service to combine them on a server.	DApp developers
F10. Some developers use factory pattern in DApps, ie, deploying a factory contract which is used to deploy new child smart contracts. Child contracts have more concentrated deployment costs, and generally have higher execution costs.	Because the factory contract must have the code of child contracts, complex child contracts lead to high deployment cost. If so, equivalent pattern may help to avoid extra deployment cost of the factory contract.	DApp developers

Abbreviations: DApps, decentralized applications.

Cryptocurrencies, for example, Bitcoin, are applications based on P2P networks as well. From 2009, more and more cryptocurrencies are issued, which leads to the growth of blockchain technology, especially public blockchain technology. According to CoinMarketCap,[60] nowadays, there are over 2000 cryptocurrencies in the world.

Few researches are about blockchain-based DApps but these DApps have great influence on the blockchains on which they run. Some work has been performed to help developers develop Blockchain-based DApps.[61]

9 | CONCLUSION AND FUTURE WORK

The scale of DApp market is rapidly growing and has reached billions of dollars. In this study, we conducted a systematic descriptive analysis of 995 DApps over Ethereum. We analyzed the popularity of DApps, the development practices of DApps, and the cost of running DApps. Our findings provided valuable implications for different stakeholders in the ecosystem of blockchain-based DApps, including end users, DApp developers, and blockchain vendors.

This paper mainly focused on the descriptive analysis of DApps' transaction data. In our future work, we plan to study how to develop a high-quality DApp by deeply investigating the source code of DApps. Some questions needed to be further explored, such as how to build a DApp project, how to keep synchronization on and off the chain, and how to improve the throughput of DApps. The answer of these questions can directly improve the development of DApps and benefit millions of users.

ORCID

Kaidong Wu https://orcid.org/0000-0001-9818-6592
Yun Ma https://orcid.org/0000-0001-7866-4075

REFERENCES

1. Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. 2008.
2. Zheng Z, Xie S, Dai HN, Chen X, Wang H. Blockchain challenges and opportunities: a survey. *Int J Web Grid Serv*. 2018;14(4):352-375.
3. BitTorrent. 2019. https://www.bittorrent.com/
4. Bitmessage. 2019. https://bitmessage.org/
5. Popcorn time. 2019. https://popcorn-time.ch/
6. Ethereum project. 2019. https://www.ethereum.org/
7. Dapp.com. 2018 DApp market report. 2019. https://www.dapp.com/article/annual-dapp-market-report-2018
8. DApp survey results 2019. 2019. https://medium.com/fluence-network/dapp-survey-results-2019-a04373db6452
9. DApp developers survey results. 2019. https://hackernoon.com/dapp-developers-survey-results-1c763901e756
10. Gervais A, Karame GO, Wüst K, Glykantzis V, Ritzdorf H, Capkun S. On the security and performance of proof of work blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS); 2016; Vienna, Austria.
11. Zheng P, Zheng Z, Luo X, Chen X, Liu X. A detailed and real-time performance monitoring framework for blockchain systems. Paper presented at: 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP); 2018; Gothenburg, Sweden.
12. Wang J, Wang H. Monoxide: scale out blockchains with asynchronous consensus zones. In: Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation (NSDI); 2019; Boston, MA.
13. Luu L, Chu DH, Olickel H, Saxena P, Hobor A. Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS); 2016; Vienna, Austria.
14. Kosba A, Miller A, Shi E, Wen Z, Papamanthou C. Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. Paper presented at: 2016 IEEE Symposium on Security and Privacy (SP); 2016; San Jose, CA.
15. Szabo N. Smart contracts. Virtual School; 1994.
16. Szabo N. Smart contracts: building blocks for digital markets. *Extropy: J Transhumanist Thought*. 1996.
17. State of the DApps. 2019. https://www.stateofthedapps.com
18. Ethereum (ETH) blockchain explorer. 2019. https://etherscan.io/
19. Android Apps on Google Play. 2019. https://play.google.com/store/apps
20. Chen W, Zheng Z, Cui J, Ngai E, Zheng P, Zhou Y. Detecting Ponzi schemes on Ethereum: towards healthier blockchain technology. In: Proceedings of the 2018 World Wide Web Conference (WWW); 2018; Lyon, France.
21. R. The R project for statistical computing. 2019. https://www.r-project.org/
22. IDEX - decentralized Ethereum asset exchange. 2019. https://idex.market/
23. King S, Nadal S. Ppcoin: peer-to-peer crypto-currency with proof-of-stake. selfpublished paper. August 2012;19.

24. Larimer D. *Delegated Proof-of-Stake (DPOS)*. Bitshare whitepaper. 2014.
25. Gilad Y, Hemo R, Micali S, Vlachos G, Zeldovich N. Algorand: scaling Byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles (SOSP); 2017; Shanghai, China.
26. Barger A, Manevich Y, Mandler B, Bortnikov V, Laventman G, Chockler G. Scalable communication middleware for permissioned distributed ledgers. In: Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR); 2017; Haifa, Israel.
27. Zhang F, Eyal I, Escriva R, Juels A, Van Renesse R. REM: resource-efficient mining for blockchains. In: Proceedings of the 26th USENIX Conference on Security Symposium (SEC); 2017; Vancouver, Canada.
28. Chen T, Li X, Wang Y, et al. An adaptive gas cost mechanism for Ethereum to defend against under-priced DoS attacks. In: *Information Security Practice and Experience: 13th International Conference, ISPEC 2017, Melbourne, VIC, Australia, December 13-15, 2017, Proceedings*. Cham, Switzerland: Springer International Publishing AG; 2017:3-24.
29. Liao CF, Cheng CJ, Chen K, Lai CH, Chiu T, Wu-Lee C. Toward a service platform for developing smart contracts on blockchain in BDD and TDD styles. Paper presented at: 2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA); 2017; Kanazawa, Japan.
30. Porru S, Pinna A, Marchesi M, Tonelli R. Blockchain-oriented software engineering: challenges and new directions. Paper presented at: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C); 2017; Buenos Aires, Argentina.
31. Coblenz M. Obsidian: a safer blockchain programming language. In: Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C); 2017; Buenos Aires, Argentina.
32. Klems M, Eberhardt J, Tai S, Härtlein S, Buchholz S, Tidjani A. Trustless intermediation in blockchain-based decentralized service marketplaces. In: *Service-Oriented Computing: 15th International Conference, ICSOC 2017, Malaga, Spain, November 13-16, 2017, Proceedings*. Cham, Switzerland: Springer International Publishing AG; 2017:731-739.
33. Shae Z, Tsai JJP. On the design of a blockchain platform for clinical trial and precision medicine. Paper presented at: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS); 2017; Atlanta, GA.
34. Ruta M, Scioscia F, Ieva S, Capurso G, Di Sciascio E. Supply chain object discovery with semantic-enhanced blockchain. In: Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SenSys); 2017; Delft, The Netherlands.
35. Bahri L, Girdzijauskas S. When trust saves energy-a reference framework for proof-of-trust (PoT) blockchains. Paper presented at: Companion Proceedings of The Web Conference (WWW); 2018; Lyon, France.
36. Liu B, Yu XL, Chen S, Xu X, Zhu L. Blockchain based data integrity service framework for IoT data. Paper presented at: 2017 IEEE International Conference on Web Services (ICWS); 2017; Honolulu, HI.
37. Nguyen HL, Ignat CL, Perrin O. Trusternity: auditing transparent log server with blockchain. Paper presented at: Companion Proceedings of the The Web Conference (WWW); 2018; Lyon, France.
38. Xu J, Wang S, Bhargava B, Yang F. A blockchain-enabled trustless crowd-intelligence ecosystem on mobile edge computing. *IEEE Trans Ind Inform*. 2019;15:3538-3547.
39. Ron D, Shamir A. Quantitative analysis of the full bitcoin transaction graph. In: *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2013:6-24.
40. Meiklejohn S, Pomarole M, Jordan G, et al. A fistful of bitcoins: characterizing payments among men with no names. In: Proceedings of the 2013 Conference on Internet Measurement Conference (IMC); 2013; Barcelona, Spain.
41. Chen T, Zhu Y, Li Z, et al. Understanding Ethereum via graph analysis. Paper presented at: IEEE INFOCOM 2018-IEEE Conference on Computer Communications; 2018; Honolulu, HI.
42. Chen T, Li X, Luo X, Zhang X. Under-optimized smart contracts devour your money. Paper presented at:2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER); 2017; Klagenfurt, Austria.
43. Chang J, Gao B, Xiao H, Sun J, Yang Z. sCompile: critical path identification and analysis for smart contracts. arXiv preprint arXiv:1808.00624. 2018.
44. Grech N, Brent L, Scholz B, Smaragdakis Y. Gigahorse: thorough, declarative decompilation of smart contracts. In: Proceedings of the 41st International Conference on Software Engineering (ICSE); 2019. Montreal, Canada.
45. What is a DApp? Decentralized application on the blockchain. 2019. https://blockchainhub.net/decentralized-applications-dapps/
46. Locasto ME, Parekh JJ, Keromytis AD, Stolfo SJ. Towards collaborative security and P2P intrusion detection. In: Proceedings from The 6th Annual IEEE SMC Information Assurance Workshop; 2005; West Point, NY.
47. Zhang X, Chen S, Sandhu R. Enhancing data authenticity and integrity in P2P systems. *IEEE Internet Comput*. 2005;9(6):42-49.
48. Cooper C, Dyer M, Handley AJ. The flip Markov chain and a randomising P2P protocol. In: Proceedings of the 28th ACM Symposium on Principles of Distributed Computing (PODC); 2009; Calgary, Canada.
49. Liang J, Kumar R, Xi Y, Ross KW. Pollution in P2P file sharing systems. In: Proceedings IEEE 24th Annual Joint Conference of The IEEE Computer and Communications Societies; 2005; Miami, FL.
50. Lian Q, Zhang Z, Yang M, Zhao BY, Dai Y, Li X. An empirical study of collusion behavior in the maze P2P file-sharing system. Paper presented at: 27th International Conference on Distributed Computing Systems (ICDCS); 2007; Toronto, Canada.
51. Saroiu S, Gummadi PK, Gribble SD. Measurement study of peer-to-peer file sharing systems. In: Proceedings Volume 4673, Multimedia Computing and Networking; 2001; San Jose, CA.
52. Pouwelse J, Garbacki P, Epema D, Sips H. The bittorrent P2P file-sharing system: measurements and analysis. In: *Peer-to-Peer Systems IV: 4th International Workshop, IPTPS 2005, Ithaca, NY, USA, February 24-25, 2005. Revised Selected Papers*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2005:205-216.

53. Loo BT, Hellerstein JM, Huebsch R, Shenker S, Stoica I. Enhancing P2P file-sharing with an internet-scale query processor. In: Proceedings of the 30th International Conference on Very Large Data Bases-Volume 30 (VLDB Endowment); 2004; Toronto, Canada.
54. Nurminen JK, Meyn AJR, Jalonen E, Raivio Y, Marrero RG. P2P media streaming with HTML5 and WebRTC. Paper presented at: 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS); 2013; Turin, Italy.
55. Amad M, Meddahi A, Vanwormhoudt G. A self-adaptive ALM architecture for P2P media streaming. Paper presented at: 2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS); 2015; Paris, France.
56. Liu CL, Wang CY, Wei HY. Cross-layer mobile chord P2P protocol design for VANET. *Int J Ad Hoc Ubiquitous Comput*. 2010;6(3):150-163.
57. Napster: home. 2019. https://us.napster.com/
58. Briola D, Micucci D, Mariani L. A platform for P2P agent-based collaborative applications. *Softw: Pract Exper*. 2019;49(3):549-558.
59. Friedman R, Portnoy A. A generic decentralized trust management framework. *Softw: Pract Exper*. 2015;45(4):435-454.
60. CoinMarketCap. All cryptocurrencies. 2019. https://coinmarketcap.com/all/views/all/
61. Dong Z, Lee YC, Zomaya AY. Proofware: proof of useful work blockchain consensus protocol for decentralized applications. arXiv preprint arXiv:1903.09276. 2019.

How to cite this article: Wu K, Ma Y, Huang G, Liu X. A first look at blockchain-based decentralized applications. *Softw: Pract Exper*. 2019;1–18. https://doi.org/10.1002/spe.2751