Analyzing Ethereum's Contract Topology

Lucianna Kiffer
Northeastern University
lkiffer@ccs.neu.edu

Dave Levin
University of Maryland
dml@cs.umd.edu

Alan Mislove
Northeastern University
amislove@ccs.neu.edu

ABSTRACT

Ethereum is the second most valuable cryptocurrency today, with a current market cap of over $68B. What sets Ethereum apart from other cryptocurrencies is that it uses the blockchain to not only store a record of transactions, but also smart contracts and a history of calls made to those contracts. Thus, Ethereum represents a new form of distributed system: one where users can implement contracts that can provide functionality such as voting protocols, crowdfunding projects, betting agreements, and many more. However, despite the massive investment, little is known about how contracts in Ethereum are actually created and used.

In this paper, we examine how contracts in Ethereum are created, and how users and contracts interact with one another. We modify the `geth` client to log all such interactions, and find that contracts today are three times more likely to be created by *other contracts* than they are by users, and that over 60% of contracts have *never* been interacted with. Additionally, we obtain the bytecode of all contracts and look for similarity; we find that less than 10% of user-created contracts are unique, and less than 1% of contract-created contracts are so. Clustering the contracts based on code similarity reveals even further similarity. These results indicate that there is substantial code re-use in Ethereum, suggesting that bugs in such contracts could have wide-spread impact on the Ethereum user population.

ACM Reference Format:
Lucianna Kiffer, Dave Levin, and Alan Mislove. 2018. Analyzing Ethereum's Contract Topology. In *2018 Internet Measurement Conference (IMC '18), October 31-November 2, 2018, Boston, MA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3278532.3278575

1 INTRODUCTION

Ethereum is a novel cryptocurrency that uses a blockchain not only to store a record of transactions, but also to store user-generated programs called *smart contracts* and a history of calls made to those contracts. Smart contracts expand the functionality and usability of blockchain-based cryptocurrencies; users have developed contracts to implement voting protocols, funding programs, gambling, and many more. As a result, Ethereum is currently the second most valuable cryptocurrency with a market cap of over 68 billion dollars as of May 2018, and various other cryptocurrencies are incorporating smart contracts.

IMC '18, October 31-November 2, 2018, Boston, MA, USA

ACM ISBN 978-1-4503-5619-0/18/10…$15.00
https://doi.org/10.1145/3278532.3278575

Smart contracts introduce a new layer of complexity and interactivity to the already-diverse ecosystem of cryptocurrencies. At the bottommost layer, most cryptocurrencies are built on top of a peer-to-peer communication substrate (e.g., the Kademlia [15] distributed hash table in Ethereum), which have received significant study [17]. At the topmost layer, users interact with one another, perform transactions, and make large protocol decisions such as when to fork—these dynamics, as well, have been studied extensively [13]. Smart contracts sit somewhere in the middle: users create them and interact with them (e.g., by contributing to a "go-fund-me" contract), but they also serve as an intermediate layer, as contracts can be built to rely on other contracts.

Although smart contracts are in many ways the essence of Ethereum and other contract-based cryptocurrencies, surprisingly little is known about them empirically. Many open questions remain: How many distinct contracts are there? How long-lived are they? To what extent are users creating wholly new contracts versus copying existing code? How many contracts rely on the availability of a given contract?

To answer these questions and more, this paper initiates the study of Ethereum's *smart contract topology*. Viewing contracts as nodes and contract calls as edges, we are able to measure the importance, connectivity, and central points of failure of Ethereum's smart contracts. To do so, we collect the bytecode of all contracts published to the Ethereum blockchain during its first 5 million blocks (almost three years), and also collect a trace of an instrumented Ethereum virtual machine (`geth`) to log all interactions between users and contracts. We apply the results of this analysis to answer two broad questions:

1. How is Ethereum being used? With the rampant speculation in the cryptocurrency markets (e.g., Bitcoin), one may wonder how this has impacted Ethereum. We find that while Ethereum's market cap and exchange rate has grown over 1,000-fold during our measurement period, the fraction of activity on Ethereum that involves contracts has remained relatively constant (roughly 1/3 of all transactions are destined to contracts, rather than users). However, we do observe evidence of a number of attacks on the Ethereum platform, often exploiting mis-priced virtual machine operations that were later adjusted.

2. How are contracts being used? Given the high level of interaction with contracts, we examine how these contracts are created and how they interact. Surprisingly, we find that roughly three times more contracts today are created by *other contracts* than are created by users; many of these contracts are sub-currency contracts, or cryptocurries built on top of Ethereum. Additionally, we find that roughly 60% of all contracts that have been created have never been interacted with, suggesting there exists significant amounts of dormant code and currency. Finally, we find extremely high levels of code re-use and code similarity: the 1.2M user-created

contracts can be reduced to 5,877 contract "clusters" that have highly-similar bytecode. This high level of code re-use suggests that bugs or vulnerabilities in these contracts could easily impact thousands more; such vulnerabilities have been discovered in the past, and have led to hundreds of millions of dollars in lost value [12, 23].

2 BACKGROUND

We now provide background on Ethereum, how it works, and the dataset that we collected for this paper.

2.1 Ethereum

Ethereum is a blockchain-based distributed system, much in the spirit of related systems such as Bitcoin. However, it has a number of unique features that make it distinct. Most notably, unlike other blockchain-based systems that primarily serve as a virtual currency, Ethereum serves both as a virtual currency *and* a distributed virtual machine.[1] Users can upload *code* to Ethereum—called *contracts*—that are run in a deterministic fashion by all participants; each contract has its own memory state and currency balance. The creating user and others can later invoke (or *call*) these contracts, in response to which contracts can transfer funds or call and even create other contracts.

Internally, Ethereum is based on *accounts*, which can either be controlled by a public/private key pair (called *external accounts*; these are for users) or controlled by code (called *contract accounts*). Both types of accounts have *storage* (essentially a random-access memory that maps 256-bit addresses to 256-bit values) and a *credit balance* of Ether (the unit of currency in Ethereum). Accounts can interact via messages that can come in multiple forms; one of these is *transactions* that appear on the blockchain. Each transaction contains a payload (which may be empty) as well as an amount of Ether (which may be 0). The payload and Ether are used differently in different contexts, as described below.

Ethereum has experienced a number of attacks on different aspects of its distributed virtual machine during its short lifetime. These include attacks on coding errors in popular contracts (such as the DAO attack that allowed an attacker to steal $50M and resulted in a hard fork [13]), attacks on incorrect settings of the cost of virtual machine opcodes (such as the "Spurious Dragon" and "Tangerine Whistle" hard forks that changed the price of certain operations [2, 3]), and contracts whose popularity inadvertently led to denial-of-service-like behavior (such as the massive popularity of "Crypto Kitties" that slowed the entire Ethereum network [4]). As we will see later in the paper, responding to each of these attacks has significantly altered the behavior of many contracts.

2.2 Opcodes

The Ethereum VM supports over 100 different instructions called *opcodes* [26]. Each opcode has a different cost to execute, based on the amount of resources that it requires. Many of these provide typical low-level features like mathematical operations (e.g., `ADD`), memory loads and stores (e.g., `MLOAD`), and other bookkeeping (e.g., `GETPC`). However, there are a few opcodes that we use when trying to understand the Ethereum contract ecosystem, which we describe below. First, contracts are created and destroyed using one of three mechanisms:

Users create new contracts by sending a transaction to a special `0x0` address; the payload of this transaction is the bytecode of the new contract.

`CREATE` is an opcode that allows a contract to create another contract. One of the arguments is the raw bytecode that the new contract should use.

`SELFDESTRUCT` is an opcode that allows a contract to self-destruct. One of the arguments is a destination address to transfer the remaining balance of the contract to.

Second, there are a few ways in which one account can call another contract's code:

Users can call contracts by sending a transaction to the contract's address, with the function being called and the inputs to the function in the payload. This call may result in messages to other contracts in the form of the opcodes below.

`CALL` is an opcode that allows a contract A to call contract C, and C's code runs in the context of C. In other words, C's storage is used, so C's code can read/write from C's storage.

`CALLCODE` is an opcode that allows a contract A to call contract C, and C's code runs in the context of A. Thus, A's storage is used for all reads and writes. Unfortunately, `CALLCODE` had a bug in its implementation and has been deprecated in favor of `DELEGATECALL`.

`DELEGATECALL` is a newer version of `CALLCODE` that fixes `CALLCODE` and was released in 2015.

`STATICCALL` is the same as `CALL`, except that it does not allow any state modifications during execution of the callee (or any sub-calls). It was introduced in 2017 in response to certain reentrancy attacks.

2.3 Dataset

One feature of Ethereum is that all contract state and code is public; this is necessary to implement the distributed virtual machine. This allows us to obtain data on how Ethereum is actually used by running a modified version of the `geth` Ethereum client. As part of its operation, the `geth` client will download and execute the entire history of the Ethereum distributed virtual machine; we modify this client to log all of the operations above (contract creations, deletions, and calls) that were successfully executed. We ran this client up through the 5,000,000th block, covering a time period from inception (in 2015) through January 30, 2018. Additionally, we query the `geth` client for all transaction data including the bytecode of contracts. We collected the ETH exchange rates from `coinmarket.com` (and verified it against other exchanges including Coinbase and CoinDesk).

3 HOW IS ETHEREUM BEING USED?

We begin our analysis by looking at the overall usage of Ethereum over time. Figure 1 plots the number of Ethereum transactions/day, the number of transactions/day that are to contract addresses, the number of transactions/day that fail to execute successfully, and the price of Ether (ETH) per USD over the course of the first 5M blocks. We make a number of observations.

[1] Technically, Ethereum offers a stack-based virtual machine and does not use registers.

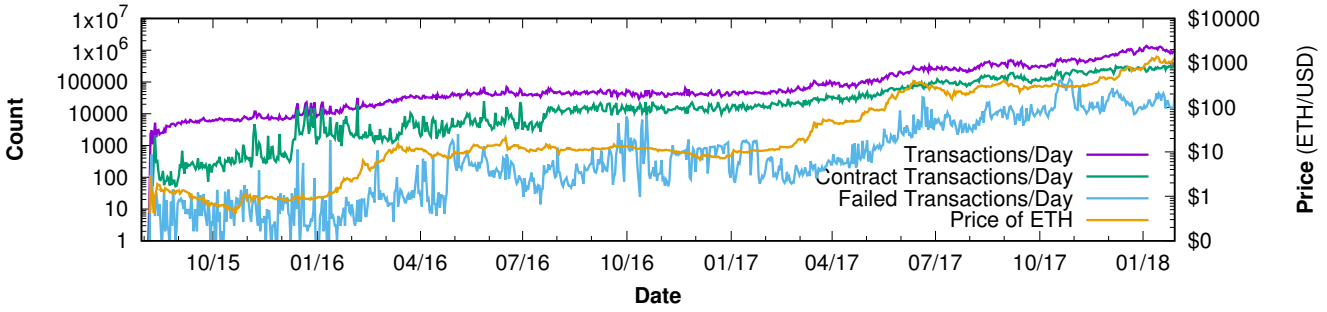

Figure 1: Number of transactions per day (left y axis), and the price of ETH in USD (right y axis), over our 3.5-year study period. Note that both y-axes are in log scale. A dramatic increase in both transactions and price can easily be observed.

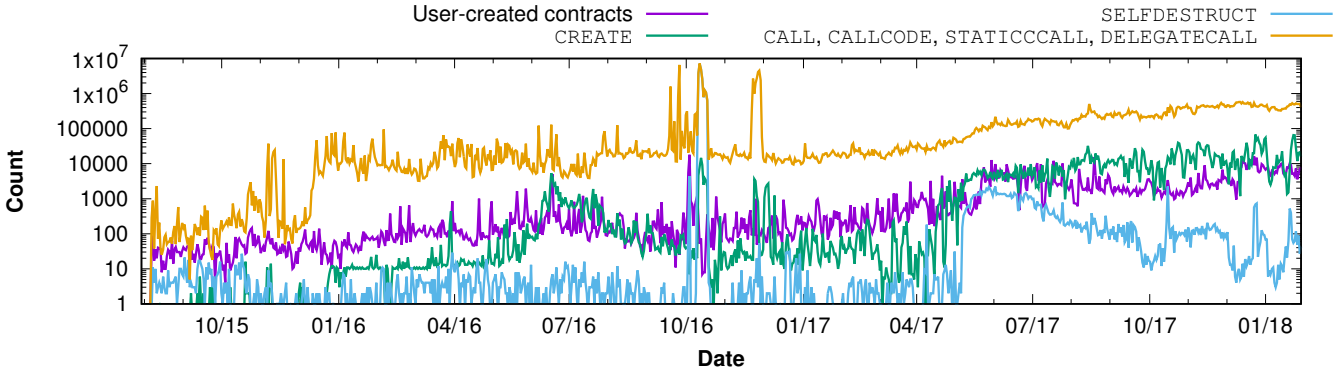

Figure 2: The number of contracts created by users, the number created by `CREATE`, the number of contracts that are destroyed via `SELFDESTRUCT`, and the number of contract calls (`CALL`, `CALLCODE`, `DELEGATECALL`, `STATICCALL`) over time.

First, we can observe that as with other blockchain-based systems like Bitcoin, Ethereum's currency has observed a dramatic increase in value over the past few years. For example, the price of ETH at the beginning of 2017 was roughly $10; a year later, it was over $1,400. Since this rapid increase, the price of ETH has returned to be closer to the $500 range, though the price remains extremely volatile.

Second, in parallel with this increase in price, we can observe that Ethereum's usage—as measured by the number of transactions in the system—has increased exponentially as well. Over calendar year 2017, the average number of transactions per day increased from roughly 40K to over 1M; despite the recent drop in price, the number of transactions per day has recently remained stable.

Third, recall that transactions can either be to users (typically simple transfers of funds, akin to Bitcoin) or to contracts (i.e., using Ethereum's virtual machine features). If we focus on the number of transactions that are to contracts, we find that over the past two years, these have consistently represented roughly 1/3 of all transactions (e.g., in early 2017, there were roughly 12K transactions to contracts per day; this grew to roughly 300K at the end of 2017). Thus, largely independent of the wild volatility of ETH price and resulting currency speculation, a significant fraction of the activity in Ethereum is using the advanced contract features.

Fourth, if we focus on the number of failed transactions we observe per day, we can see that it typically is roughly 3–4 orders of magnitude smaller than the number of total transactions (i.e., roughly 0.01%–0.1% of transactions typically fail). However, we can observe two periods during which the number of failed transactions is significantly higher than this: during a short period in October 2016 and during another, longer, period in November 2017. Digging into these periods, we find that both were caused by external events: in October 2016, an attacker exploited the mis-pricing of a number of operations to execute a denial-of-service attack on the network; this was fixed with a hard fork that raised the price of these operations. In November 2017, the launch of the "Crypto Kitties" contracts caused a massive increase in usage of Ethereum; this resulted in many failed transactions initially. These results indicate that contract-centric activity can serve as a lens into broader events in the Ethereum ecosystem.

To summarize, we see that a significant fraction of the activity in Ethereum is using its smart contracts, and that exogenous events can be reflected in how they are used. In the next section, we dive into these contracts and explore how the are used and inter-related.

4 HOW ARE CONTRACTS BEING USED?

We now take a closer look at the Ethereum contracts, focusing first on their lifecycles before examining how they are related to each other.

4.1 Contract life cycle

We begin by examining how contracts are created, deleted, and called. Figure 2 plots the number of contracts created by users, the number created by CREATE, the number of contracts that are destroyed via SELFDESTRUCT, and the number of contract calls (CALL, CALLCODE, DELEGATECALL, STATICCALL) over time. *First*, we can immediately observe that while, historically, the number of user-created contracts have tended to dominate contract-created contracts, that trend reversed in early 2017. In fact, today, there are over 1.2M user-created contracts in existence while there are over 3.4M contract-created contracts! In looking into why there are so many contract-created contracts, especially after April 2017, we found that many of these are "token contracts", or custom currencies created on top of Ethereum that use contract-created contracts to implement certain functionality. We see a similar change at that time with the usage of SELFDESTRUCT as well, likely by the same set of contracts.

Second, we can observe the October 2016 denial of service attack more clearly here. During that time, malicious contracts were repeatedly calling SELFDESTRUCT resulting in a expensive call (in terms of CPU time) that was underpriced at the time; the usage of this opcode jumped from under 10 calls per day across all contracts to over 4M! With the hard fork that followed to address the issue, the usage quickly dropped back to its normal rate.

Third, we can observe that there are a tremendous number of contract calls throughout the lifetime of Ethereum. In fact, there are roughly 1–2 orders of magnitude more contract calls than than contract creations, suggesting that the average number of calls per contract is significant. In the next section, we take a closer look at how these contracts interact via calls.

Finally, we take a look at *who* is creating the contracts. We define the set of contracts C_0 as those contracts that were created directly by users; similarly, we define other sets of contracts C_i as those contracts that were created by a contract in C_{i-1}. Occasionally for simplicity we will refer to $C_{>0}$, which represents all contracts *not* in C_0 (i.e., all contract-created contracts). Table 1 presents a breakdown of the number of contracts in each set. Surprisingly, we find that the "tree" of contract creations is quite shallow: C_3 is

		Unique number of		
Set	**Size**	**Creators**	**Bytecode**	**Opcodes**
C_0	1,208,174	43,397	125,177	96,378
C_1	3,490,092	3,930	2,368	2,325
C_2	11,253	2,544	72	72
C_3	1	1	1	1

Table 1: Breakdown of the number of contracts in each set, along with the number of unique creators (addresses of users or contracts), unique bytecodes, and unique opcodes (bytecodes ignoring opcode arguments).

		C_0	$C_{>0}$
Number of contracts		1,208,174	3,501,334
Number of contracts receiving at least one	**Transactions**	407,403	193
	CALL	72,335	1,482,835
	STATICCALL	2	0
	CALLCODE	59	3
	DELEGATECALL	4,755	23,855

Table 2: Number of unique contracts that were ever the recipient of a CALL or STATICCALL message, or have their code called via CALLCODE or DELEGATECALL.

the final level of the tree, and it only contains a single contract.[2] Additionally, if we look at the number of creators, we find that a very small minority of the contracts and users are responsible for most contract creation. For example, set C_1 are contracts created by contracts in C_0; while there are over 3.4M contracts in C_1 and 1.2M contracts in C_0, only 3.9K of the contracts in C_0 are responsible for creating all 3.4M contracts in C_1.

4.2 Contract interaction graph

Thus far, we have examined how contracts are created. Now we turn to examine how contracts are *used* by focusing on the call graph. Recall from Figure 2 that there are a large number of transactions sent to contracts every day; Table 2 breaks all of those calls down by (a) the type of contract that is receiving the transaction (the columns), and (b) the type of transaction that was sent (the rows). We can immediately observe that most contracts in both C_0 and $C_{>0}$ are never interacted with, either by being the recipient of a transaction or via a call! For C_0, less than 40% of contracts show any interaction; for $C_{>0}$, less than 42% do. While these contracts could of course be called in the future, the large fraction of them that have remained dormant is nevertheless surprising.

4.3 Contract equality

We now turn to examine how similar contracts are to one another. Recall that when a user creates a contract, it is given a unique contract address, even if some other user had previously created a contract with exactly the same bytecode. Thus we begin by examining how often users create exactly the same contract, which we call *bytecode-level equality*. From the geth client, we are able to obtain the bytecode for all contracts that have not executed SELFDESTRUCT. Table 1 shows the number of unique bytecodes at each level of the creation graph. We can observe that *many* users and contracts appear to be creating the same contracts (across 1.2M user-created contracts, only 125K distinct bytecodes exist; across 3.4M contract-created contracts, only 2K distinct bytecodes exist). This high level of code reuse for user-created contracts suggests users are obtaining their code from a small set of locations, including that if bugs exist in these contracts, the effects could be widespread.

While examining this code, we observed that many contracts appear to be the same, only with a few different arguments to certain opcodes (typically destination addresses for currency transfers). We

[2]The only C_3 contract is a result of a *token factory* contract (a contract that creates token contracts).

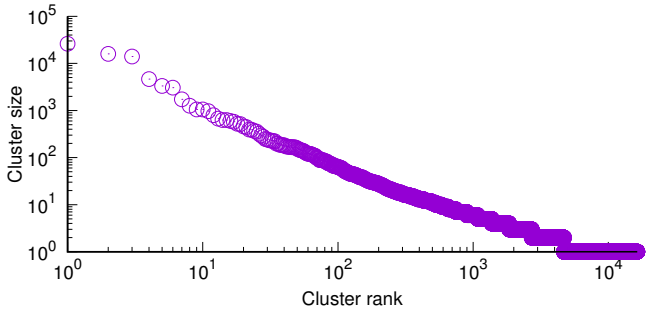

Figure 3: The cluster sizes for user-generated contracts follows a long-tail distribution. Of the 125,177 bytecode-unique user-generated contracts, there are only 16,373 clusters, the largest comprising 26,144 contracts and 11,678 clusters of size one.

therefore define two contracts to be equivalent under *opcode-level equality* if their bytecode is identical when ignoring the opcode arguments. Table 1 shows the number of unique contracts using opcode-level equality in the final column. We find that about 25% of the bytecode-level unique user-created contracts are not opcode-level unique, indicating even further code reuse.

4.4 Contract similarity

Next, we seek to understand the extent to which users reuse portions of code from one contract in another. Such code reuse can have ramifications on the reliability of the contract ecosystem: a bug or vulnerability in one contract could potentially be copied elsewhere. Moreover, in blockchain systems like Ethereum, smart contract code can never be changed as it is part of the permanent blockchain state; instead, entirely new contracts need to be created and migrated to.

To measure code reuse, we compute the *similarity* between two contracts as follows: First, we disassemble all contracts and strip out all arguments to opcodes (but maintain the order in which the opcodes appear). Then, we compute the frequency of all n-grams in both contracts (we use $n = 5$, but found similar results for larger values of n). This yields, for each contract, a hypervector in the high-dimensional space of 5-grams of Ethereum opcodes. To compare two contracts, we compute the cosine similarity between their respective hypervectors, resulting in a number between zero (completely dissimilar) and one (identical).

We spot-checked dozens of contracts and found that, when two contracts have a score of 0.90 or higher, they exhibit extensive code reuse. We also verified that this similarity score is highly transitive: for three contracts a, b, and c, if $\text{sim}(a, b) \geq 0.90$ and $\text{sim}(b, c) \geq 0.90$, then $\text{sim}(a, c) \geq 0.90$ over 88.8% of the time.

We use this similarity score to cluster contracts together. Specifically, we cluster together any two contracts with score 0.90 or higher. We note that a contract belongs to the first cluster it matches to. We run our clustering algorithm on the set of all user-generated contracts, and on the set of all contract-generated contracts. Figure 3 shows the distribution of cluster sizes for user-generated contracts. Of the 125,177 total contracts, we find that there are only 16,373 clusters, indicating extensive code reuse throughout the Ethereum smart contract ecosystem. We find the distribution to exhibit a long tail; the largest cluster comprises 20.9% (26,144) of all user-generated contracts; the top five clusters constitute 51.1%. Conversely, there is a very long tail, with 11,678 clusters of size one. We spot checked the top three largest clusters against the available source code[3] and found that the largest cluster is made up of token contracts, the second largest cluster is made up of contracts involved in the DDoS attack of October 2016, and the third largest cluster is made up of wallet contracts.

For contract-generated contracts, we also saw high levels of code reuse. They, too, exhibit a long-tail distribution, with a largest cluster of 361 contracts. Code reuse among these forms of contracts is not quite as rampant as with user-generated contracts; of the 2,440 contracts, we found 694 clusters, the largest five of which comprise 37.5% of all such contracts. We also compared the source code of the three largest clusters of the contract-generated contracts and found that all three clusters are made up of token contracts.

4.5 Summary

The results in this section have a consistent message: that Ethereum's smart contract ecosystem has a considerable lack of diversity. Most contracts reuse code extensively, and there are few creators compared to the number of overall contracts. It remains to be seen whether this lack of diversity is endemic to smart contracts (or Ethereum itself), or if it is merely a reflection of the relative youth of smart contracts as a whole—perhaps as new contracts and modes of interactions are developed, we will see an increase in diversity.

In the mean time, the high levels of code reuse represent a potential threat to the security and reliability. Ethereum has been subject to high-profile bugs that have led to hard forks in the blockchain [13] or resulted in over $170 million worth of Ether being frozen [1]; like with DNS's use of multiple implementations, having multiple implementations of core contract functionality would introduce greater defense-in-depth to Ethereum.

5 RELATED WORK

There has been extensive work towards developing an empirical understanding of various aspects of the cryptocurrency ecosystem. Early work in this space inspected the transactions taking place on Bitcoin's blockchain including transaction patterns [5, 22], properties of repeated subgraphs [16] and hypergraphs [21], and the UTXO set [10]. Work focusing on privacy and anonymity of Bitcoin transactions have looked at transaction history [6], address clustering [19] and mixing services [18] to de-anonymize addresses. Others have focused on transactions related to scams, Ponzi schemes and Ransomware [8, 24, 25] and the impact of what kind of data is stored as part of Bitcoin transactions [14].

Other work has studied Bitcoin's peer-to-peer network—including how information propagates [9], how networks react to partitions [13], how to leverage the peer-to-peer topology to infer which nodes are more *influential* or linked to mining pools [17],

[3]Contract source code is available at `etherscan.io` and can be verified against the bytecode.

and other studies on decentralization in Bitcoin and Ethereum's peer-to-peer networks [11].

Conversely, in this paper, we explore a unique and growing aspect of the cryptocurrency ecosystem: smart contracts. The most closely related work to ours involve analysis of the kinds of contracts being written. Norvill et al. clustered 998 Ethereum contracts whose source code were available on the block explorer etherscan.io at the time of their study [20]. They looked at the frequency of the most common words in the code and clusters based on context triggered piecewise hashes of the bytecode of these contracts. Bartoletti and Pompianu studied 811 Ethereum contracts with available source code [7] and categorized them into 5 categories: financial, notary, game, wallet and library. They look at how many transactions relate to each category and find that around 66% of transactions to contracts at the time were to financial categorized contracts. They did a similar analysis for Bitcoin transactions which use a scripting language and encode some metadata in transactions and also found that financial categorized transactions were the most popular. Their analysis of the Ethereum contracts included manually inspecting them to identify different design patterns.

Compared to both of these studies, we perform our analysis over a much larger scale of contracts. We are not limited to contracts for which there is publicly available code; we measure contract similarity based on n-grams of the decompiled bytecode for all unique bytecodes, allowing us to examine similarity at a much larger scale. As a result, we believe our work to be an important first step towards developing a more comprehensive understanding of the smart contract ecosystem.

6 CONCLUSION

Smart contracts are a fundamental addition to cryptocurrencies; just as it is important to study the peer-to-peer network topology and user-to-user transaction activity, we argue that it is equally important to study how users interact with smart contracts. In this paper, we have initiated the study of Ethereum's smart contracts at-scale by investigating its smart contract topology. Our initial findings indicate high levels of contract activity (largely independent of price), but low levels of contract diversity: most contracts are direct- or near-copies of other contracts. While this is likely a driving force behind Ethereum's success (copying another's contract is an easy way to start using the system), it also represents a potential risk, if buggy or vulnerable code were to be copied.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. This research was supported in part by NSF grants CNS-1409191, CNS-1409249, CNS-1564143, CCF-1422715 and CCF-1535929.

REFERENCES

[1] Another parity wallet hack explained. https://medium.com/@Pr0Ger/another-parity-wallet-hack-explained-847ca46a2e1c.
[2] Spurious dragon hard fork. https://blog.ethereum.org/2016/11/18/hard-fork-no-4-spurious-dragon/, November 2016.
[3] Tangerine whistle. https://blog.ethereum.org/2016/10/18/faq-upcoming-ethereum-hard-fork/, October 2016.
[4] Cryptokitties craze slows down transactions on ethereum. http://www.bbc.com/news/technology-42237162, December 2017.
[5] L. Anderson, R. Holz, A. Ponomarev, P. Rimba, and I. Weber. New kids on the block: an analysis of modern blockchains. *arXiv preprint arXiv:1606.06530*, 2016.
[6] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer, 2013.
[7] M. Bartoletti and L. Pompianu. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *International Conference on Financial Cryptography and Data Security*, pages 494–509. Springer, 2017.
[8] H. Basil Al Jawaheri, M. Al Sabah, and Y. Boshmaf. Measurement and analysis of bitcoin transactions of ransomware. In *Qatar Foundation Annual Research Conference Proceedings*, volume 2018, page ICTPD1026. HBKU Press Qatar, 2018.
[9] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *International Conference on Peer-to-Peer Computing (P2P)*, pages 1–10. IEEE, 2013.
[10] S. Delgado-Segura, C. Pérez-Sola, G. Navarro-Arribas, and J. Herrera-Joancomartı. Analysis of the bitcoin utxo set.
[11] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. G. Sirer. Decentralization in bitcoin and ethereum networks. *arXiv preprint arXiv:1801.03998*, 2018.
[12] A. Hertig. $160 million stuck: Can parity still shake up ethereum? https://www.coindesk.com/startup-lost-160-million-still-wants-shake-ethereum/.
[13] L. Kiffer, D. Levin, and A. Mislove. Stick a fork in it: Analyzing the ethereum network partition. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 94–100. ACM, 2017.
[14] R. Matzutt, J. Hiller, M. Henze, J. H. Ziegeldorf, D. Müllmann, O. Hohlfeld, and K. Wehrle. A quantitative analysis of the impact of arbitrary blockchain content on bitcoin. In *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC). Springer*, 2018.
[15] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.
[16] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Internet Measurement Conference*, pages 127–140. ACM, 2013.
[17] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee. Discovering bitcoin's public topology and influential nodes. *et al.*, 2015.
[18] M. Moser. Anonymity of bitcoin transactions: An analysis of mixing services. In *Münster Bitcoin Conference (MBC)*, 2013.
[19] T. Neudecker and H. Hartenstein. Could network information facilitate address clustering in bitcoin? In *International Conference on Financial Cryptography and Data Security*, pages 155–169. Springer, 2017.
[20] R. Norvill, B. B. F. Pontiveros, R. State, I. Awan, and A. Cullen. Automated labeling of unknown contracts in ethereum. In *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*, pages 1–6. IEEE, 2017.
[21] S. Ranshous, C. A. Joslyn, S. Kreyling, K. Nowak, N. F. Samatova, C. L. West, and S. Winters. Exchange pattern mining in the bitcoin transaction directed hypergraph. In *International Conference on Financial Cryptography and Data Security*, pages 248–263. Springer, 2017.
[22] D. Ron and A. Shamir. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.
[23] D. Siegal. Understanding the dao attack. https://www.coindesk.com/understanding-dao-hack-journalists/.
[24] M. Vasek and T. Moore. There's no free lunch, even using bitcoin: Tracking the popularity and profits of virtual currency scams. In *International conference on financial cryptography and data security*, pages 44–61. Springer, 2015.
[25] M. Vasek and T. Moore. Analyzing the bitcoin ponzi scheme ecosystem. In *Bitcoin Workshop*, 2018.
[26] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.