PlaTIBART: a Platform for Transactive IoT Blockchain Applications with Repeatable Testing

Michael Walker
Vanderbilt University
michael.a.walker.1@vanderbilt.edu

Abhishek Dubey
Vanderbilt University
abhishek.dubey@vanderbilt.edu

Aron Laszka
Vanderbilt University
laszka.aron@gmail.com

Douglas C. Schmidt
Vanderbilt University
d.schmidt@vanderbilt.edu

ABSTRACT

With the advent of blockchain-enabled IoT applications, there is increased need for related software patterns, middleware concepts, and testing practices to ensure adequate quality and productivity. IoT and blockchain each provide different design goals, concepts, and practices that must be integrated, including the distributed actor model and fault tolerance from IoT and transactive information integrity over untrustworthy sources from blockchain. Both IoT and blockchain are emerging technologies and both lack codified patterns and practices for development of applications when combined. This paper describes PlaTIBART, which is a platform for transactive IoT blockchain applications with repeatable testing that combines the Actor pattern (which is a commonly used model of computation in IoT) together with a custom Domain Specific Language (DSL) and test network management tools. We show how PlaTIBART has been applied to develop, test, and analyze fault-tolerant IoT blockchain applications.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

Internet of Things, Design Patterns, Testing, Blockchains

ACM Reference format:
Michael Walker, Abhishek Dubey, Aron Laszka, and Douglas C. Schmidt. 2017. PlaTIBART: a Platform for Transactive IoT Blockchain Applications with Repeatable Testing. In *Proceedings of 4th Workshop on Middleware and Applications for the Internet of Things, Las Vegas, USA, December 2017 (MIDDLEWARE2017),* 7 pages.

MIDDLEWARE2017, December 2017, Las Vegas, USA

ACM ISBN 123-4567-24-567/08/06...$15.00
https://doi.org/10.475/123_4

https://doi.org/10.475/123_4

1 INTRODUCTION

Interest in—and commercial adoption of—blockchain technology has increased in recent years [7]. For example, blockchain adoption in the financial industry has yielded market capitalization surpassing $75 billion USD [2] for Bitcoin and $36 billion USD for Ethereum [4]. One reason for this growth stems from blockchains' combination of existing technologies to enable the interoperation of non-trusted parties in a decentralized, cryptographically secure, and immutable ecosystem, without the need of a trusted central authority.

During roughly the same time, the increased proliferation of IoT devices has motivated the need for transactional integrity due to the transition of IoT devices from just being smart-sensors to being active participants that impact their environment via communication, decision making, and physical actuation. These abilities require transactional integrity to provide auditing of actions made by potentially untrusted networked 3rd party IoT devices. The demand for transactional integrity in IoT devices that simultaneously leverage blockain features (such as decentralization, cryptographic security, and immutability) has motivated research on creating transactive IoT blockchain applications [12, 13, 24].

Blockchain deployments (and specifically Ethereum, which is focus in this paper due to its large installed base and its powerful smart sontract language) are generally managed via programs that have different modes in which they can operate. They broadly fall into Command-Line Interfaces (CLI), RPC APIs, or creating Graphical Interfaces via the use of HTML pages and JavaScript code [5]. These interfaces provide standard means to either run Ethereum applications within the clients themselves or to interface other applications with the Ethereum clients.

In practice, however, the existing blockchain deployment interfaces lack built-in fault tolerance, most notably for either network communication errors or application execution faults. Moreover, Ethereum clients are deployed manually since no official manager exists for them. As a result, developers can—and do [8]—lose all of their Ether (Ethereum's digital currency) due to insecure client configurations. This problem is compounded by the fact that Ethereum's clients do not

warn of this risk within their built-in help feature and instead rely upon online documentation to warn developers.

Addressing this problem requires patterns and tools that enable the deployment of blockchain clients in a repeatable and systematic way. This requirement becomes even more important when integrating IoT blockchain applications (ITBAs). The IoT component of ITBAs adds other requirements atop traditional blockchain applications due to their interactions with the physical environment and increased privacy concerns, *e.g.*, preventing leakage of personal data, such as energy usage that would reveal a user's activity patterns in their home [22].

Moreover, ITBAs may not only communicate over the blockchain, but may also use off-blockchain communications via TCP/IP or other networking protocols for the following reasons:

- There are interactions with the physical environment that might require communication with sensors and/or actuators. For example, a user's smart-meter might communicate wirelessly with their smart-car's battery to activate charging based on current energy production/cost considerations.
- The distributed ledger (which makes an immutable record of transactions in blockchain) is public, so it is common to only include information within transactions that can safely be stored publicly. In particular, if some or all data from a transaction must be kept secret for privacy or any other reasons, the transaction can instead contain the meta-data and a cryptographic hash of the secret data. Private information must therefore be communicated off-blockchain, while still preserving integrity by storing meta-data and hash information on the blockchain ledger.
- Management tasks, such as updates, monitoring, calibration, debugging, or auditing, may require off-blockchain communication (with possible on-blockchain components for logging). Currently these management tasks are done manually in conventional blockchain ecosystems. Similar to the need for a systematic means of deploying apps in a blockchain network, there is a need to systematically configure the network topology between all components of ITBAs.

This paper presents the structure and functionality of PlaTIBART, which is a *Platform for Transactive IoT Blockchain Applications with Repeatable Testing* that provides a set of tools and techniques for enhancing the development, deployment, execution, management, and testing of ITBAs. In particular, we describe a pattern for developing ITBAs, a Domain Specific Language (DSL) for defining a private blockchain deployment network, Actor components upon which the application can be deployed and tested, a tool using these DSL models to manage deployment networks in a reproducible test environment, and interfaces that provide fault tolerance via an application of the *Observer* pattern.

The remainder of this paper is organized as follows: Section 2 explains the system model underlying PlaTIBART and describes the scenario of transactive energy used in this paper to motivate the need for ITBAs; Section 3 reviews the current state-of-the-art regarding IoT and blockchain integration; Section 4 illustrates our proposed ITBA architecture and shows how we use the *Actor* pattern to construct our solution, the DSL we created, and the network manager script we created to generate test networks for ITBAs; Section 5 examines our experimental testbed configuration and analyzes our results; Section 6 summarzies lessons learned while implementing our proposed architecture; and Section 7 presents concluding remarks and outlines future work.

2 SYSTEM MODEL

This section explains the system model underlying PlaTIBART and describes the use case scenario of transactive energy we use in this paper to motivate the need for ITBAs.

Based on our experience developing decentralized apps (DApps) for blockchain ecosystems [19, 36], three key capabilities must be provided for DApps to function effectively in an ITBA ecosystem: traditional IoT computations and interactions must be supported, information must be sorted robustly in a distributed database, and a system-wide accepted sequential log of events must be provided. Each requirement can be delegated to a separate layer in a three-tiered architecture. The first tier is the IoT Middleware layer that facilitates communication between networked devices, which can be addressed by existing IoT middleware, such as RIAPS [21]. The second tier is a distributed database layer (which is not the focus of this paper). The third tier is a sequential log of events layer, which can be solved by blockchain integration.

PlaTIBART leverages the Actor model [26] to integrate these three layers. Each layer is composed of components that accomplish their designated layer dependent tasks. These components are then combined into a single actor that can interact with each layer and other actors in the network, as described in Section 4.

2.1 Case Study: Transactive Energy System

Transactive energy systems (TES) have emerged in response to the shift in the power industry away from centralized, monolithic business models characterized by bulk generation and one-way delivery, toward a decentralized model in which end users play a more active role in both production and consumption [14, 29]. The GridWise Architecture Council defines TES as "a system of economic and control mechanisms that allows the dynamic balance of supply and demand across the entire electrical infrastructure using value as a key operational parameter" [29].

In this paper, we consider a class of TES that operates in a grid-connected mode, meaning the local electric network is connected to a Distribution System Operator (DSO) that provides electricity when the demand is greater than what the local-network can generate. The main actors are the consumers, which are comprised primarily of residential loads, and prosumers who operate distributed energy resources,

such as rooftop solar batteries or flexible loads capable of demand/response. Additionally, the DSO manages the grid connection of the network. Such installations are equipped with an advanced metering infrastructure consisting of TE-enabled smart meters. Examples of such installations include the Brooklyn Microgrid Project [3] and the Sterling Ranch learning community [16]. A key component of TES is a transaction management platform (TMP), which handles market clearing functions in a way that balances supply and demand in a local market.

3 ANALYSIS OF STATE-OF-THE-ART

This section reviews the state-of-the-art in IoT and blockchain integration, focusing on testing. Prior work [15] has shown that IoT and blockchain can be integrated, allowing peers to interact in a trustless, auditable manner via the use of blockchain as a resilient, decentralized, and peer-to-peer ledger. Work has also been done on the topics of security and privacy of IoT and Blockchain integrations [18, 31]. Beyond that, work has focused on formal verification of smart contracts [25] and how to write smart-contracts "defensively" [17] to avoid exceptions when multiple contracts interact. The current state-of-the-art with respect to testing, however, is lacking because blockchains are infrequently tested at-scale in a systematic and repeatable manner, so we focus on that below.

This demonstrates the maturity of Blockchain profiling tools, which further increases the already high privacy requirements on development of Blockchain enabled IoT devices and networks. Additionally, this shows that considerations need to be taken into account to consider all possible avenues in which personal information is disclosed to prevent profiling.

3.1 Testing IoT Blockchain Systems

Popular blockchain ecosystems, such as Bitcoin and Ethereum, suffer from design limitations that prevent their direct application to IoT. In particular, transaction-confirmation times are relatively long (around tens of minutes) and variable on public blockchain networks, due largely to their proof-of-work algorithms. Likewise, IoT devices have limited processing power and storage capabilities, which must be accounted for and tested [10] to ensure constraints are met.

Prior work on testing of IoT blockchain systems generally fall into two categories: (1) their test implementation has a single client and one or more smart contracts or (2) they focus purely on theoretical aspects and discuss future work implementing a test example. For example, Beck et. al [11] discuss their implementation, but apparently (it is not discussed in detail) only use a single client, two smart contracts, and no additional transactions on the ledger. Conversely, Simic et. al [34] present a purely theoretical paper, where they discuss IoT and blockchain powered healthcare at a high level without addressing privacy or any of the many other significant implementation difficulties.

3.2 Testing Repeatability

The importance of integration and regression testing in software development has been well-known for over 20 years [9, 28, 32]. Integration and regression testing of distributed systems has been improved via network emulation testbeds, such as DETERLAB [30] and Emulab [33], which provided mechanisms to repeatably deploy and test a distributed system for both integration and regression testing.

Testing ITBAs incurs additional difficulties that standard IoT applications do not face. For example, there is a completely separate network for each component of the Actor in an ITBA: the IoT middleware/application layer, possibly a distributed datebase layer, and the blockchain layer. We focus on the IoT middleware/application layer and blockchain layer in this paper. Therefore, testing requires, per Actor, that we must run both the Actor's IoT middleware/application code and a blockchain client instance. This pairing brings a wide range of conditions that must be planned for, tracked, corrected, and tested.

Some examples of what must be tested include (1) the order of actor/blockchain client starting, (2) whether all actors should be started before processing on either the IoT and/or blockchain network starts, and (3) what detection, and recovery mechanisms will be implemented to account for lost messages between the blockchain client and the actor, the actor losing a message, and transactions being lost on the blockchain. A testing environment for ITBAs thus needs to repeatably create networks and network conditions to address these conditions. Section 4 describes how the PlaTIBART architecture enables the building of such test networks.

4 OUR PROPOSED ARCHITECTURE

PlaTIBART applies a pattern for creating repeatable test network deployments of IoT/blockchain applications that combines a Domain Specific Language (DSL) to define the network topology and settings, a Python program leveraging the Fabric API to manage the test network, and the RIAPS middleware [21] to facilitate communication between nodes on the network. Each of these components is described below.

4.1 Application Platform

The *Resilient Information Architecture Platform* for Smart Grid (RIAPS) [21] is the application platform PlaTIBART uses to implement our case-study example described in Section 2.1. RIAPS provides actor and component based abstraction, as well as support for deploying algorithms on devices across the network [1] and solves problems collaboratively by providing micro-second level time synchronization [21], failure based reconfiguration [19], and group creation and coordination services (still under active development), in addition to the services described in [27]. It is capable of handling different communications and running implemented algorithms in real-time.

Mike: This is 3rd time we use RIAPS, but first spelling out, is that okay?

[1] RIAPS uses ZeroMQ [23] and Cap'n Proto [35] to manage the communication layer.

4.2 Actor Pattern

Each application client in the network is implemented as an actor with two main components: (1) a wrapper class specific to the role the actor is given and (2) a geth client, the reference client for Ethereum[2]. Figure 1 shows a small network of 5 actors, indicated by an ellipse around a Wrapper and geth client pair, and the networking connections between each actor's components. Geth clients communicate exclusively via on-blockchain means, *i.e.*, the geth client of each actor communicates directly with its associated wrapper, and the wrapper communicates directly with other wrappers via an off-blockchain channel, such as TCP P2P communications.

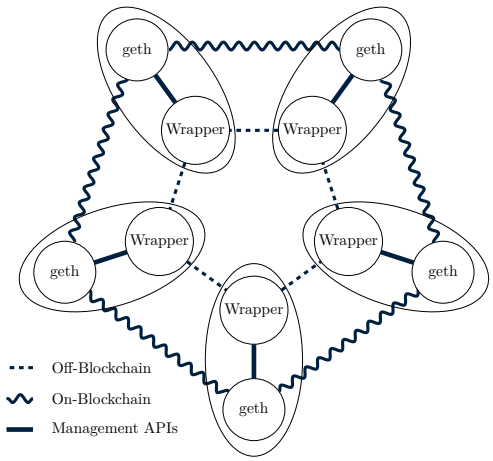

Figure 1: Sample Actor component network, an Actor is a geth client and a Wrapper.

4.3 Fault Tolerance

A key benefit of decoupling the blockchain client and the wrapper into two components of an actor is enhanced fault tolerance around transaction loss compared with tightly coupled solutions. Specifically, it allows the wrapper to not only monitor the blockchain client, but also shut down and restart the client as needed.

This design allows the wrapper component to ensure that if any known or discovered faults arise from defects in the blockchain software the wrapper can at least attempt to recover. For example, in our test networked described in Section 5.1, we have encountered faults where transactions are never mined [1] until a client is restarted. Other types of faults, such as those related to an actor's communication with other components of the network, are handled by other middleware solutions, such as RIAPS.

PlaTIBART applies the *Observer* pattern to notify the wrapper of the occurrence of events, such as faults and other blockchain-related conditions. This notification is accomplished by a separate thread within the wrapper that monitors its paired geth client for new events, such as completed transactions, or potential faults. This thread then notifies registered callback(s) when target events occur. For example, if the geth client becomes unresponsive or transactions appear to have stalled, then registered callback method(s) are called to notify the wrapper. .

[2]https://github.com/ethereum/go-ethereum/wiki/geth

4.4 Domain Specific Language

PlaTIBART's DSL defines the roles that different clients in our network have based on the *Actor* pattern. This DSL model implements a correct-by-construction design, thereby allowing for a verification stage on the model to check for internal consistency before any deployment is attempted. This verification prevent inconsistencies, such as two clients requesting the same port on the same host.

Figure 2 shows an example of our DSL, which specifies a full network configuration file for a test network. The first two lines of the configuration file contain two unique identifiers for this test network and its current version, "configurationName" and "configurationVersion" respectively. Next, it contains values specific for the creation of an Ethereum private network's Genesis block.

A Genesis block in Ethereum is the first block in a blockchain and has special properties, such as not having a predecessor and being able to declare accounts that already have balances before any mining or transactions begin. The "chainID" is a unique positive integer identifying which Blockchain the test network is using; 1 through 4 are public Ethereum blockchains of varying production/testing phases.

Next, "difficulty" indicates how hard it is to mine a block and "gasLimit" is the maximum difficulty of a transaction based on length in bytes of the data and other Ethereum runtime values. The "balance" is the starting balance that we allocate to each client's starting account upon creation of the network, which eliminates the situation where clients cannot begin transactions to request assets before any mining has begun. Lastly, the "clients" represent the actual nodes in our network.

Clients in the DSL represent the individual actors in our network, comprised of a geth client and a RIAPs instance using a Wrapper interface. The geth client has two interfaces and TCP ports associated with it: one for incoming Blockchain connections, and one for administration.

In the network defined in Figure 2, we allocate each client a pair of ports on a private subnet, and for each new client type we offset the port by 1,000. This means that in the example network defined by Figure 2, Prosumers use ports 9,000-9,029, DSO uses ports 10,000 and 10,001, and "miners" use ports 11,000 and 11,001. This design limits the total number of clients to 1,000, though this limit can be removed by assigning unique start ports to each client type.

5 EMPIRICAL RESULTS

This section examines our experimental testbed configuration and analyzes our experiment's results.

	2 Prosumers		5 Prosumers		10 Prosumers		15 Prosumers		20 Prosumers	
Testing Stage	**Avg**	**Std Dev**	**Avg**	**Std Dev**	**Avg**	**Std Dev**	**Avg**	**Std Dev**	**Avg**	**Std Dev**
Clients Create	8.592	0.172	14.927	0.097	24.194	0.160	33.597	0.194	42.642	0.285
Miners Create	2.785	0.011	2.845	0.066	2.755	0.008	2.773	0.042	2.754	0.029
Blockchain Make	0.069	0.001	0.070	0.001	0.071	0.002	0.072	0.004	0.070	0.000
Blockchain Create	0.233	0.010	0.230	0.020	0.218	0.018	0.220	0.009	0.228	0.015
Distribute to Clients	2.348	0.012	3.372	0.064	5.058	0.063	6.729	0.111	8.455	0.061
Distribute to Miners	0.680	0.017	0.675	0.009	0.687	0.016	0.668	0.021	0.664	0.023
Full Network Created	14.731	0.157	22.142	0.188	33.008	0.217	44.081	0.137	54.836	0.337
Miner Start	0.800	0.018	0.806	0.018	0.801	0.015	0.805	0.015	0.811	0.021
Clients Start	2.773	0.019	3.921	0.085	4.815	0.099	6.337	0.132	7.485	0.258
Network Connect	0.504	0.008	0.932	0.011	1.634	0.009	2.401	0.040	3.071	0.031
Network Stop	4.421	0.034	5.558	0.034	5.968	0.085	6.506	0.088	10.326	0.249
Network Delete	5.332	0.058	5.288	0.047	5.290	0.067	5.297	0.027	5.446	0.072

Table 1: Average Time (Seconds) and Standard Deviation of 5 Tests for Each Variation of Number of Prosumer Clients

```
{"configurationName":"test network a001",
"configurationVersion":"1",
"chainId": 15,
"difficulty": 100000,
"gasLimit": 200000000000000000,
"balance": 400000000000000000000000000,
"genesisBlockOutFile":"genesis-data.json",
"clients": {
    "startPort": 9000,
    "prosumer": {
        "count": 15,
        "protocol": "rpc",
        "hosts": [
            "10.4.209.25",
            "10.4.209.26",
            "10.4.209.27",
            "10.4.209.28" ] },
    "dso": {
        "count": 1,
        "protocol": "rpc",
        "hosts": [
            "10.4.209.29" ] },
    "miner": {
        "count": 1,
        "protocol": "rpc",
        "hosts": [
            "10.4.209.30" ] }
     }
}
```

Figure 2: Sample DSL model.

5.1 Experimental Testbed Configuration

To test our proposed solution we implemented the test network defined in Figure 2. This was installed on a private cloud instance hosted by the university. We ran our tests on 6 virtual hosts each with: 4GB RAM, 40GB Hard Drive space, running Ubuntu 16.04.02, and gigabit networking.

For these tests we implemented Wrappers for both DSO and Prosumer clients in python. Each Wrapper had 1 geth client associated with it. We used used the Network Manager tool we wrote to create, start, shutdown, and delete the test network. We manually paired each Wrapper with its geth client's IP address and port, which in the future we will integrate into the network manager's capabilities.

With our custom written Wrappers and managed test network we were able to simulate a day's worth of Transactive Energy trading between Actors. We timed, through the use of the "time" Linux command, the entire process of and each step involved in creating a test network, which are: Clients Create, Miners Create, Blockchain Make, Blockchain Create, Distribute to Clients, and Distribute to Miners. Additionally, we also timed the steps required for starting and connecting the geth instance of each "clients" ("prosumer" & "dso") to the geth client of each "miner." Currently, this star-network is the only network topology we support, but plan to expand these options in the future. Note: miner(s) are treated as a special case of "clients" and have their own unique set of Network Manager commands.

5.2 Analysis of Results

After running our tests, we found that our standard deviation for each testing phase to be small (largest being 0.09% of the time taken) and the average time to either remain relatively static or to scale linearly, with one exception, in relation to the number of clients (2, 5, 10, 15, 20 prosumers + 1 DSO). The test phases that stayed relatively static are: Miners Create, Blockchain Make, Blockchain Create, Distribute to Miners, Miners Start, and Network Delete. The test phases that scaled with increase in number of prosumers are: Clients Create, Distribute to Clients, Full Network Created, Clients Start, Network Connect, and Network Stop. Other than Network Stop, which appears exponential but might become linear at greater network sizes, all of the scaling increases are linear (Std Dev ¡ 0.065) after dividing the avg time increase by the difference in number of clients.

6 LESSONS LEARNED

During the implementation of our initial PlaTIBART prototype, we learned many lessons related to integrating IoT and blockchain. The three main categories of lessons we learned involved documentation deficiencies, buggy behavior of the Ethereum geth client, and limitations of both Ethereum management APIs and the Solidity smart contract language.

The official documentation for Ethereum is lacking in many key areas, such as organization, completeness, lack of meaningful examples, and clarity on best practices or security warnings. Here are some examples that demonstrate this:

- Ethereum does not maintain its own documentation, instead linking to an outside resource maintained by volunteers from the Ethereum Community.
- The only official documentation is a FAQ on the main page, and the wikis in Ethereum's various source code repositories.
- Important side effects of a management API call are only found listed under another method's documentation.
- The official wiki does not highlight the fact that if management APIs are made available on an interface, then they are public by default. This problem has already been reported several times on the bug reporting page, due to lost Ether.

There are also programmatic bugs with Ethereum's reference client implementation, geth. While building and evaluating our test network, we experienced new transactions that were not mined regardless of how many new blocks being mined, Ether available to the client, or any other obvious cause.

Similar issues have been reported frequently on the public bug-reporting/tracking system about others attempting to setup private networks [1]. As of writing this paper, however, there is no solution other than to restart the geth client. This issue is addressable, but highlights the importance of fault tolerance in individual client execution and ways to recover from faults at that level.

There are also idiosyncrasies of the Ethereum management APIs that are not well documented. An example is the polling mechanism that clients use to see if transactions occurred that meet certain search criteria, which are called filters. The problem is that created filters are set to an undefined timer and will simply cease to work if not used "for a while" [6]. This quote, however, does not come from the description of the method for creating the filter. Instead, it is on a secondary method, eth_uninstallFilter(), which is never referenced directly or indirectly from the original method, eth_newFilter().

There are also limits to the Solidity smart contract language that must be accounted for in early planning stages of development. For example, the language currently does not support floating-point numbers. Moreover, all values must be converted to a specific binary representation for submission as a transaction. These limitations prevent—or dramatically increase the complexity (and therefore computational cost)—of advanced mathematical computations on-blockchain, which yields more off-blockchain processing and communication.

7 CONCLUDING REMARKS

This paper describes how PlaTIBART applies the *Actor* pattern with DSL-driven test network management software and component creation to enable the development of resilient, fault-tolerant IoT-blockchain applications and middleware. We employed PlaTIBART to dynamically deploy and manage test blockchain networks of varying sizes based on DSL configuration files. We also defined APIs for monitoring and recovering from faults that standard blockchain applications were unable to recover from. This capability provides the means for fully integrated regression testing of blockchain applications, which is a novel contribution.

PlaTIBART currently uses Ethereum as its blockchain implementation. For example, our DSL has Ethereum-specific required settings, such as "chainId" and "gasLimit." Future versions of PlaTIBART will refactor these requirements so that other blockchain platforms, such as Hyperledger, can be substituted seamlessly. Other areas of future work focus on formal verification of internal-consistency of a configuration file and a means of defining incremental adjustments to a test network through the DSL. Likewise, we are developing network management tools that help to simplify and automate the network topology for both the overall test framework instance and which Actor components are paired.

REFERENCES

[1] [n. d.]. Sometimes, transactions disappear from txpool rather than being mined into the next block Issue #14893 ethereum/go-ethereum. https://github.com/ethereum/go-ethereum/issues/14893. ([n. d.]). (Accessed on 09/06/2017).

[2] 2017. Bitcoin (BTC) price, charts, market cap, and other metrics — CoinMarketCap. https://coinmarketcap.com/currencies/bitcoin/. (08 2017). (Accessed on 08/30/2017).

[3] 2017. Brooklyn Microgrid. (2017). http://brooklynmicrogrid.com/

[4] 2017. Ethereum (ETH) $381.84 (3.83%) — CoinMarketCap. https://coinmarketcap.com/currencies/ethereum/. (08 2017). (Accessed on 08/30/2017).

[5] 2017. Interfaces — Ethereum Frontier Guide. https://ethereum.gitbooks.io/frontier-guide/content/interfaces.html. (2017). (Accessed on 08/30/2017).

[6] 2017. JSON RPC ethereum/wiki Wiki GitHub. https://github.com/ethereum/wiki/wiki/JSON-RPC. (2017). (Accessed on 08/28/2017).

[7] 2017. The Truth About Blockchain. https://hbr.org/2017/01/the-truth-about-blockchain. (01 2017). (Accessed on 08/30/2017).

[8] 2017. use RPC API personal_sendTransaction lost coin Issue #14901 ethereum/go-ethereum. https://github.com/ethereum/go-ethereum/issues/14901. (08 2017). (Accessed on 08/30/2017).

[9] Hiralal Agrawal, Joseph Robert Horgan, Edward W Krauser, and Saul A London. 1993. Incremental regression testing. In *Software Maintenance, 1993. CSM-93, Proceedings., Conference on*. IEEE, 348–357.

[10] Ahmed Banafa. 2017. IoT and Blockchain Convergence: Benefits and Challenges - IEEE Internet of Things. https://iot.ieee.org/newsletter/january-2017/iot-and-blockchain-convergence-benefits-and-challenges.html. (2017). (Accessed on 08/31/2017).

[11] Roman Beck, Jacob Stenum Czepluch, Nikolaj Lollike, and Simon Malone. 2016. Blockchain-the Gateway to Trust-Free Cryptographic Transactions.. In *ECIS*. ResearchPaper153.

[12] Andreas Bogner, Mathieu Chanson, and Arne Meeuw. 2016. A Decentralised Sharing App Running a Smart Contract on the Ethereum Blockchain. In *Proceedings of the 6th International*

Conference on the Internet of Things (IoT'16). ACM, New York, NY, USA, 177–178. https://doi.org/10.1145/2991561.2998465

[13] Francesco Buccafurri, Gianluca Lax, Serena Nicolazzo, and Antonino Nocera. 2017. Overcoming Limits of Blockchain for IoT Applications. In *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES '17)*. ACM, New York, NY, USA, Article 26, 6 pages. https://doi.org/10.1145/3098954.3098983

[14] E. Cazalet, P. De Marini, J. Price, E. Woychik, and J. Caldwell. 2016. *Transactive Energy Models*. Technical Report. National Institute of Standards Technology.

[15] Konstantinos Christidis and Michael Devetsikiotis. 2016. Blockchains and smart contracts for the internet of things. *IEEE Access* 4 (2016), 2292–2303.

[16] Sterling Ranch Development Company. 2017. The Nature of Sterling Ranch. (2017). http://sterlingranchcolorado.com/about/

[17] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. 2016. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *International Conference on Financial Cryptography and Data Security*. Springer, 79–94.

[18] Ali Dorri, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. 2017. Blockchain for IoT security and privacy: The case study of a smart home. In *Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on*. IEEE, 618–623.

[19] Abhishek Dubey, Gabor Karsai, and Subhav Pradhan. 2017. Resilience at the edge in cyber-physical systems. In *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*. IEEE, 139–146.

[20] S. Eisele, I. Mardari, A. Dubey, and G. Karsai. 2017. RIAPS: Resilient Information Architecture Platform for Decentralized Smart Systems. In *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*. 125–132. https://doi.org/10.1109/ISORC.2017.22

[21] S. Eisele, I. Mardari, A. Dubey, and G. Karsai. 2017. RIAPS: Resilient Information Architecture Platform for Decentralized Smart Systems. In *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*. 125–132. https://doi.org/10.1109/ISORC.2017.22

[22] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems* 29, 7 (2013), 1645–1660.

[23] Pieter Hintjens. 2010. ZeroMQ: The Guide. *URL http://zeromq.org* (2010).

[24] Simona Ibba, Andrea Pinna, Matteo Seu, and Filippo Eros Pani. 2017. CitySense: Blockchain-oriented Smart Cities. In *Proceedings of the XP2017 Scientific Workshops (XP '17)*. ACM, New York, NY, USA, Article 12, 5 pages. https://doi.org/10.1145/3120459.3120472

[25] Ranjit Kumaresan and Iddo Bentov. 2014. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 30–41.

[26] Edward A Lee, Stephen Neuendorffer, and Michael J Wirthlin. 2003. Actor-oriented design of embedded hardware and software systems. *Journal of circuits, systems, and computers* 12, 03 (2003), 231–260.

[27] H. Lee, S. Niddodi, A. Srivastava, and D. Bakken. 2016. Decentralized voltage stability monitoring and control in the smart grid using distributed computing architecture. In *2016 IEEE Industry Applications Society Annual Meeting*. 1–9. https://doi.org/10.1109/IAS.2016.7731871

[28] Hareton KN Leung and Lee White. 1990. A study of integration testing and software regression at the integration level. In *Software Maintenance, 1990, Proceedings., Conference on*. IEEE, 290–301.

[29] R. B. Melton. 2013. *Gridwise transactive energy framework*. Technical Report. Pacific Northwest National Laboratory.

[30] Jelena Mirkovic and Terry Benzel. 2012. Teaching cybersecurity with DeterLab. *IEEE Security & Privacy* 10, 1 (2012), 73–76.

[31] Aafaf Ouaddah, Anas Abou Elkalam, and Abdellah Ait Ouahman. 2017. Towards a novel privacy-preserving access control model based on blockchain technology in IoT. In *Europe and MENA Cooperation Advances in Information and Communication Technologies*. Springer, 523–533.

[32] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering* 27, 10 (2001), 929–948.

[33] Christos Siaterlis, Andres Perez Garcia, and Béla Genge. 2013. On the use of Emulab testbeds for scientifically rigorous experiments. *IEEE Communications Surveys & Tutorials* 15, 2 (2013), 929–942.

[34] Miloš Simić, Goran Sladić, and Branko Milosavljević. 2017. A Case Study IoT and Blockchain powered Healthcare. (06 2017).

[35] Kenton Varda. 2015. Capn Proto. (2015).

[36] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town Crier: An Authenticated Data Feed for Smart Contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 270–282. https://doi.org/10.1145/2976749.2978326