Chapter 06

Testing at Scale of IoT Blockchain Applications

Michael A. Walker

Douglas C. Schmidt

Abhishek Dubey

Abstract

Due to the ever-increasing adaptation of Blockchain technologies in the private, public, and business domains, both the use of Distributed Systems and the increased demand for their reliability has exploded recently, especially with their desired integration with Internet-of-Things devices. This has resulted in a lot of work being done in the fields of distributed system analysis and design, specifically in the areas of blockchain smart contract design and formal verification. However, the focus on formal verification methodologies has meant that less attention has been given towards more traditional testing methodologies, such as unit testing and integration testing. This includes a lack of full support by most, if not all, the major blockchain implementations for testing at scale, except on fully public test networks. This has several drawbacks, such as: 1) The inability to do repeatable testing under identical scenarios, 2) reliance upon public mining of blocks, which introduces unreasonable amounts of delay for a test driven development scenario that a private network could reduce or eliminate, and 3) the inability to design scenarios where parts of the network go down. In this chapter we discuss design, testing methodologies, and tools to allow Testing at Scale of IoT Blockchain Applications.

Keywords: Automation, Blockchain, Distributed Systems, IoT, Scalability, Testing, Testing at Scale.

Introduction

Blockchain deployments (and specifically Ethereum, which is the main focus of this chapter due to its large installed base and its powerful smart contract 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. 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—lose all of their Ether (Ethereum's digital currency) due to insecure client configurations. 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 add other requirements atop traditional blockchain applications due to their interactions with the physical environment and increased privacy concerns, e.g., thus preventing leakage of personal data, such as energy usage that would reveal a user's activity patterns in their home. Additionally, ITBAs may not only communicate over the blockchain, but may also use off-blockchain communications via TCP/IP or other networking protocols for reasons related to their operation.

In this book chapter we present Best Practices for Testing-at-Scale of Blockchain Systems making use of 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 blockchain systems and specifically 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 technology/technical terms used in the book chapter are explained wherever they appear or at the "Key Terminology & Definitions" section. Apart from regular References, additional References are included in the "References for Advance/Further reading" for the benefit of advanced readers.
Structure of the proposed book chapter

6.1. Introduction of Distributed Ledgers/Blockchain Testing Concepts

Interest in —and commercial adoption of— blockchain technology has increased in recent years [31]. For example, blockchain adoption in the financial industry has yielded market capitalization surpassing $75 billion USD [4] for Bitcoin and $36 billion USD for Ethereum [15]. Blockchain's growth, at least partially, stems from its 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. Blockchain, a specific type of Distributed Ledger, provides these features in different ways depending on implementation. However, generally blockchains work by creating a cryptographically signed chain of blocks, hence the name, that are decentralized via a consensus mechanism such as Proof-of-Work, that is not controlled by a central authority. Distributed Ledgers, which share many similarities to blockchain, do not necessarily require decentralized authority. However, for this chapter we discuss both but focus on blockchain versions of distributed ledgers due to the fully distributed non-central authority being easier to implement and manage, and therefore we assume more likely, for IoT manufacturers to integrate with. Blockchain deployments (and specifically Ethereum, which is the focus of this chapter due to its large installed base and its powerful, smart contract 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 [18]. 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 [32]—lose all their Ether (Ethereum's digital currency) due to unsecure 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.

6.2. Testing Analysis of Blockchain and IoT Systems

Blockchain systems can be subdivided into two broad categories: Turing Complete and Non-Turing Complete. This means the design of the system's contract language is either Turing Complete or it is not. The largest of each of these two categories is Bitcoin as a non-Turing Complete contract language and Ethereum as a Turing-Complete contract language. The reason this is important is because it describes the inherent design goal of language. Turing-Complete languages allow for theoretically any computation to be completed, whereas non-Turing Complete languages have a more limited instruction set that specifically limit the actions available in that language. The reason for adding these limitations to the language is to limit the functionality and therefore potential complexity of code put onto the blockchain's public ledger and executed distributedly. Non-Turing Complete contract languages are easier to analyze and predict runtime behavior, results, and potential faults. Additionally, there are blockchain/distributed ledger frameworks such as Hyperledger Fabric which do not provide a specific public blockchain for use, but instead provide tools for developing customizable blockchain/distributed ledger applications or implementations modularly.

During roughly the same time as the growth of blockchain, 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 blockchain features (such as decentralization, cryptographic

security, and immutability) has motivated research on creating transactive IoT blockchain applications [5, 7].

This requirement becomes even more important when integrating IoT blockchain applications (ITBAs). The IoT component of ITBAs add other requirements atop traditional blockchain applications due to their interactions with the physical environment and increased privacy concerns, e.g., thus preventing leakage of personal data, such as energy usage that would reveal a user's activity patterns in their home [16]. 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.

6.3. Desired Functionality of Testing IoT Blockchain Systems

In this section we list desired functionality of Testing IoT Blockchain Systems. Specifically, what we believe is the simplest way to

delineate progressive levels of increased testing of IoT blockchain systems. These stages, starting at the most easily achievable and becoming progressively more difficult, are: Unit Testing, Simple IoT Device Integration, Multiple IoT Device Integration, Test Driven Development, and Fully Automated Test-Driven Development.

Unit testing of software has become a standard requirement in well developed code. However, contract languages do not always include default unit-testing capability in the language or default build environment. However, the largest implementations for different categories of Blockchain solutions: Ethereum, Bitcoin, and Hyperledger Fabric all provide unit testing functionality, so any solution that does not do so should not be considered production level ready.

Beyond unit testing, the next level of desired testing of ITBAs is integration testing. Integration testing of purely software-based distributed systems provides a unique challenge due to coordination of multiple instances, networking and runtime configuration, etc. ITBAs compound this by requiring not only multiple software instances to be run for integration testing, but also require integration with the IoT component(s) of the system to verify runtime characteristics, hardware and software compatibility, etc. Therefore, we've decided to split the stage of testing with IoT devices into two sub-stages: one where integration testing is only done with one device, and then into a second stage where multiple devices are integrated into testing. This division provides a cleaner progression of desirability for analysis of testing progress.

The next level of desired testing ITBA systems is continuous integration. Continuous integration, like unit testing and integration testing, are commonplace in software development now. However, the adoption of these practices is less dependent upon the core blockchain, or even IoT, system being used and more about the support software designed to assist in development of that specific system. Therefore, like unit testing, we suggest considering any system that doesn't yet provide continuous integration support via support libraries, tools, etc. to be non-production level ready.

6.4 Existing Shortcomings in Testing IoT Blockchain Systems

This section reviews the state-of-the-art in IoT and blockchain integration, focusing on testing. Prior work [9] 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 [12, 26]. Beyond that, work has focused on formal verification of smart contracts [20], and how to write smart contracts "defensively" [11] 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.

6.4.1. Functional vs Model-Based Declarations

Currently, as far as we can tell, PlaTIBART is the only model-based system for deploying Blockchain test networks,with Ethereum or otherwise. There are some tools, such as Nixos,[1] that provide for repeatable installation of their Linux distribution and therefore via use of the NixOps devops tool, can declaratively define deployments of private Ethereum networks. However, this still requires functional declaration of the instances to be created. The benefits of a model-based approach are that it allows much easier variation in the outputs, additionally, a model-based declaration can be modified to create the functional declaration inputs of other systems easily, thereby maintaining easy adaptability while also increasing interoperability with other tools, toolchains, and workflows.

6.4.2. Testing on Live Environments, Non-Repeatable

Blockchain systems, particularly Ethereum, focused extensively at the start on testing your smart contract code on a public, global, and non-modifiable instance of the Ethereum network they call the Test Network. Ethereum has at least added support for smart contract unit testing,

[1] https://nixos.org/

testing smart contracts in an emulator, and calling that integration testing. However, these approaches lack robustness and repeatability.

The use of a public non-blockchain, even a testing one, for development poses several potential issues for developers. Firstly, the chance of publishing content to the blockchain that is intended to be secret is a high concern in a test environment. Secondly, reliance upon a public blockchain for testing removes the ability to control the frequency, latency, and predictability, or lack thereof, of your testing environment. This is important due to the common need for tests to be faster than real-time execution speed.

The use of an emulator to do integration testing of only the smart contract component of the system lacks robustness because of several reasons. First, it doesn't use the same client as production code would. Second, it ignores the need to include the client itself in the integration testing process. Third, it focuses on the HTML/JavaScript interface of the official client, while ignoring the other interfaces that geth provides, such as the JSON RPC API.

Therefore, we believe Ethereum has issues with the design philosophy of their testing mechanisms. Additionally, we have noted previously [34] that Ethereum's documentation was incomplete and spread across multiple pages for the same APIs, and as of the date of this publication the issue still exists.

6.4.3. Lack of Defined Integration/Testing Methodologies

Unfortunately, there is currently a severe lack of support for testing Blockchain systems and software when not using the precise scenarios envisioned by the Blockchain system's creators. For instance, Ethereum doesn't have any tools, testing or otherwise, that assist in integrating the official command line client of Ethereum: geth into applications. There is an official IDE, the Remix Solidity IDE, which enables unit testing but no support for integration testing at all currently. Their focus is on unit testing their smart contracts and "integration testing" their contracts inside a separate simulator, and not the geth client and private test networks. Other Blockchain and/or Distributed Ledger technologies, such as Hyperledger Fabric, at least have unit testing support and support integration testing, but at the time of writing, they have zero documentation on it.

6.5. Platform for Transactive IoT Blockchain Applications with Repeatable Testing (PlaTIBART)

The following sections will describe the PlaTIBART architecture, components, and components.

6.5.1. System Design/Rationale

PlaTIBART architecture for creating repeatable test network deployments of IoT/blockchain applications 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[14] to facilitate communication between nodes on the network. Each of these components is described below.

6.5.2. Application Platform

The *Resilient Information Architecture Platform* for Smart Grid (RIAPS)[14] is the application platform used by PlaTIBART to implement our case-study examples.]

RIAPS provides actor and component based abstraction, as well as support for deploying algorithms on devices across the network[2] and solves problems collaboratively by providing micro-second level time synchronization[14], failure based reconfiguration[7], and group creation and coordination services (still under active development), in addition to the services described in [22]. It is capable of handling different communications and running implemented algorithms in real-time.

6.5.3. 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[3]. Figure 1 shows a small network of five actors (indicated by an ellipse

[2] RIAPS uses ZeroMQ [17] and Cap'n Proto [33] to manage the communication layer.

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

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.

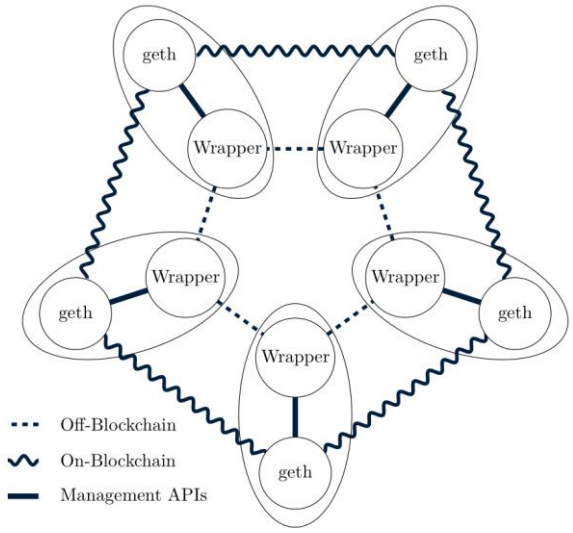

Figure 1 : Sample Actor Component Network with an Actor is a Geth Client and a Wrapper.

6.5.4. 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 Ethereum test network described in Section 6.7, we have encountered faults where transactions are never mined [32] a client is restarted. These lost transactions are problematic since they prevent a client from being able to interact with the blockchain network. 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.

6.5.5. Domain Specific Language

PlaTIBART's DSL defines the roles that different clients in our net-work 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 prevents inconsistencies, such as two clients requesting the same port on the same host.

Figure 1 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 and should not be used for creation of private networks.

Next, "difficulty" indicates how computationally 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[4], 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.

```
{" configurationNa me ":" test network a001 ",
" configurationVe rs io n ":"1" ,
" chainId ": 15 ,
" difficulty ": 100000 ,
" gasLimit ": 200000000000000000 ,
" balance ": 40000000000000000000000000 ,
" genesisBlockOutFil e ":" genesis - data . json ",
" clients ": {
        " startPort ": 9000 ,
         " prosumer ":{
                " count ": 15 ,
                 " hosts ": [ "10.4.209.25" ,
                              "10.4.209.26" ,
                               "10.4.209.27" ,
                               "10.4.209.28" ]
                }
        , " dso ": {
                " count ": 1,
                 " hosts ": [ "10.4.209.29" ]
                },
        " miner ": {
                " count ": 1,
                " hosts ": [ "10.4.209.30" ]
                }
         }
}
```

Figure 2 Sample DSL Model

[4] “balance” applies only to accounts created before a new blockchain is created. Accounts created after the blockchain, be it public or private, is created will not receive any starting balance.

Figure 2 shows how Clients are defined. 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 interface/TCP port pairs associated with it: one for incoming Blockchain connections, and one for administration and communication with RIAPs.

6.5.6. Network Manager

Based on our experience developing decentralized apps (DApps) for blockchain ecosystems [19, 34], three key capabilities are essential for DApps to function effectively in an ITBA ecosystem: traditional IoT computations and interactions should be supported, information should be robustly sorted in a distributed database, and a system-wide accepted sequential log of events should 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 [14]. The second tier is a distributed database layer. The third tier is a sequential log of events layer, which can be solved by blockchain integration. PlaTIBART provides an architecture for coordinating all these layers in a fault tolerant manner, along with tools for repeatable testing at scale. It leverages the Actor model [21] 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 6.7 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 [8, 24]. 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” [24]. In this paper, we consider a class of TES that operates in a gridconnected 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 TES-enabled smart meters. Examples of such installations include the Brooklyn Microgrid Project [6] and the Sterling Ranch learning community [10]. 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.

6.6. In-Depth Guided Walkthrough of PlaTIBART Network Manager

The goal of PlaTIBAT is to use models to design and deploy repeatable testing networks for IoT Blockchain Applications. Therefore, we present a guided walkthrough of how PlaTIBART's Network Manager allows for simple command line creation of blockchain networks, currently only Ethereum but with more to come in future revisions of the software.

The Network Manager, having file name network-manager.py, is designed to be a command line tool for eventual integration into other systems, such as automated build systems, etc. Therefore, it follows best practices of command line tools and has a built-in help menu to assist users when learning the system. Additionally, it doesn't do anything to the system that can't already be done by a series of repetitive, and potentially complicated, command line instructions. Meaning that the purpose of the Network Manager is to simplify the process of creating blockchain test networks in a repeatable and model-driven manner. The use of a model allows the command line instructions to remain the same for almost all variations of supported network designs. Currently the Network Manager only supports networks designs where each blockchain node connects to each of the bootnodes, or to each of the first class of clients. Higher level of customization in network connections is a future area of research and development. Having the same series of instructions for the Network Manager enables a series of simple commands to be written that fully automate test network creation and testing. To show this, we'll be examining a complete cycle of creating, running, stopping and deleting a test network.

6.6.1. Guided Walkthrough of Creating New Test Network

The first step in creating a test network is to delete the temporary files that can be used. The following commands delete the /new-blockchain/ directory where the blockchain is created and saved to. The remaining files are possible files that may or may not be created depending on system design. The file static-nodes.json is a list of static nodes, a possible Ethereum discovery mechanism, that could have been previously created. The new miners and clients json files are lists of newly created miner clients and standard non-miner clients. Clients are separated into two categories due to the relatively massive memory requirements for miners, 4 GB minimum to even start mining and growing from there as the blockchain grows in length, versus the relatively minor approximately 250 MB of ram used by a non-miner client note these requirements being specific to the Ethereum network and Ethereum's client: geth. The genesis-data.json file is the traditional input file that geth uses to create a new blockchain network, the Network Manager creates this as an intermediary artifact during network creation.

The first step in creating a new Ethereum test network is to create Bootnodes if your network is going to use them. We're going to assume a valid PlaTIBART model file is passed as a parameter to $1 in the

```
rm -f ./static-nodes.json
rm -f ./new-miners.json
rm -f ./new-clients.json
rm -rf ./new-blockchain/
rm -f ./genesis-data.json
```

Figure 3: Commands to Delete Network Manager Temporary Files

following command line instructions both save space and to reinforce that the Network Manager's commands don't change based on input model.

```
./network-manager.py bootnodes create --file $1 \
 --out=./static-nodes.json
```

Figure 4 How to create Bootnodes with the Network Manager

The next step in creating a new Ethereum test network is to create the clients and miners defined in the input model file. The order of these commands isn't dependent upon one another.

```
./network-manager.py clients create --file $1 \
 --out=./new-clients.json

./network-manager.py miners create --file $1 \
 --out=./new-miners.json
```

Figure 5 Creating miners and clients

Next in creating a new Ethereum test network is to make the genesis-data.json input file that allows Ethereum's geth client to create a new blockchain network. This file contains all the meta-data about the network to be created, such as staring Ether for known clients, complexity of the beginning mining calculations, and the ChainID, which prevents unrelated chains from communicating with each other. This is then fed to the local copy of geth on the host machine and creates the genesis block (first block in a blockchain).

```
./network-manager.py blockchains make --file $1 \
 --clients ./new-clients.json
```

Figure 6 Making genesis file for new test network

Here is the creation of the genesis block, which is done on the host machine's local geth client.

```
./network-manager.py blockchains create \
 --file genesis-data.json --datadir ./new-blockchain/
```

Figure 7 Creating the new Blockchain genesis block

```
./network-manager.py clients distribute --file $1 \
 --local ./new-blockchain/

./network-manager.py miners distribute --file $1 \
 --local ./new-blockchain/
```

Figure 8 Distributing genesis block to clients and miners

This newly created block contains all the model's meta data and allows pre-mining distribution to each of the miners and clients. This pre-mining distribution helps prevent a potential race-condition where a client is always trying to "catch-up" to the newest created block and never does. If the mining difficulty is set too low compared to the processing power of the system(s) hosting the miner(s) this is a possibility that can occur.

At this point, localized logic and data files can also be distributed via the Network manager to the hosts for each one of the generated clients from the model. The specifics of these files will depend upon the ITBA(s) that you are testing. This example distributed the code used in our Use-Case 1. At this point the new Blockchain, Ethereum in this example, test network is now fully created and ready for use. Some use cases, such as our Use-Case 2, will archive this network for future use, while others will make use of it as is.

6.6.2. Guided Walkthrough of Starting and Stopping Test Networks

```
./network-manager.py clients distribute --file $1 \
 --local ./components/ --subdir components/

./network-manager.py clients distribute --file $1 \
 --local ./data/ --subdir components/data/
```

Figure 9 Distributing logic code and data to each client

```
./network-manager.py miners connect --file $1
```

Figure 10 Connecting miners to each client to connect the network

```
./network-manager.py miners start --file $1

./network-manager.py clients start --file $1
```

Figure 11 Starting miner and clients

Starting the network for mining and then the processing of data and requests start by starting the miners and clients. If the miners and clients can reach a single bootnode in the bootnode network, then they should eventually sync up if the network is moderately reliable. Otherwise, if not using bootnodes, the miners will need connected to the clients manually.

Stopping the entire network can be done step-by-step by using the above commands, substituting "start" with "stop". Alternatively, you can use the Network Manager's network options to stop the entire network at once. Starting the entire network also works, but it was explained in detail above for clarity.

```
./network-manager.py network stop --file $1
```

Figure 12 Network Manager stopping entire network

Deleting the network will delete all files created by the setup process on all clients, miners, and bootnodes in the network, but will not delete the host machine's generated files, if those are to be kept separately.

```
./network-manager.py network delete --file $1
```

Figure 13 Network Manager deleting entire network

6.7. Example Use-Case 1: Transactive Energy

Transactive Energy Systems (TES) have emerged in resonse 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 [8, 24]. 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" [24].

6.7.1. Sample Problem

In this section, we consider a class of TES that operates in a gridconnected 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 TES-enabled smart meters. Examples of such installations include the Brooklyn Microgrid Project [6] and the Sterling Ranch learning community [10]. 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.

6.7.2. In-Depth Guided Walkthrough

To test PlaTIBART we implemented a solution to the Transactive Energy case study and deployed it to the test network defined in Figure 2. This network was installed on a private cloud instance hosted at Vanderbilt 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 a custom smart contract and wrappers for both Smart Grid distribution system operators (DSO) and prosumer clients in Python. Each wrapper had one geth client associated with it. We used PlaTIBART's network manager tool outlined in Section 6.5.6 and commands detailed in Sections 6.6.1 and 6.6.2 to create, start, shutdown, and delete the test network. We manually paired each wrapper with its geth client's IP address and port (in future work this is to be integrated and automated into the network manager's capabilities). Using our custom written wrappers, smart contract, and managed test network we simulated a day's worth of transactive energy trading between actors. Via the Linux "time" command we measured each step needed in the entire process to create a test network, including Clients Create, Miners Create, Blockchain Make, Blockchain Create, Distribute to Clients, and Distribute to Miners. We also measured the steps required to start and connect the geth instance for each "clients" ("prosumer" and "DSO") to the geth client of each "miner." Currently, this star-network is the only network topology supported by PlaTIBART, but we will expand the supported topologies in the future.

6.7.3. System Output and Analysis

After running our tests, described above, we found the standard deviation for each testing phase was small (the largest being 0.09% of the time taken). Likewise, the average time either remained relatively static, or scaled linearly, in relation to the number of clients (2, 5, 10, 15, 20 prosumers + 1 DSO + 1 miner).

The test phases that remained relatively static included: 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 were: Clients Create, Distribute to Clients, Full Network Created, Clients Start, Network Connect, and Network Stop. The scaling increases were linear (Std Dev < 0.065) after dividing the average time increase by the difference in number of clients.

The results of our experiments indicate that there exists high consistency and predictability of managing PlaTIBART-managed blockchain test networks. These results help build confidence that PlaTIBART's approach to creating repeatable testing networks for IoT blockchain applications scales well, which is important to encourage adoption by IoT system developers.

6.8. Example Use-Case 2: Blockchain/Distributed Systems Education

As we discussed in An Elastic Platform for Large-scale Assessment of Software Assignments for MOOCs (EPLASAM), there are significant challenges presented when attempting to scale software assignments for use in MOOCs[35]. Attempting to scale distributed system, specifically ITBA, software assignments presents additional challenges beyond those of traditional software assignments. Even ignoring the need for physical IoT hardware to test code on, the need for: private repeatable Blockchain networks, easily adjustable network designs, and ease of use of network setup by both instructors and staff, but also learners, becomes crucial for individual assessment.

6.8.1. Sample Problem

The use of PlaTIBART does not provide a complete solution, as described in EPLASM, for MOOC scalable assessment, or even just testing, of ITBAs. However, it does provide the design philosophy, tools, and methods that enable repeatable individual assessment of ITBAs and/or ITBA components. In our classes at both Vanderbilt University and Youngstown State University, we have made use of PlaTIBART to assist in the creation assignments that assessed the Blockchain components of ITBAs. Additionally, we were able to leverage PlaTIBART to provision 25x IoT clusters (Cisco router, 4x Raspberry Pi 3B+, ethernet cabling, and an IoT Electronics kit) for a series of IoT lectures and workshops held in partnership with Youngstown State University, the Youngstown Business Incubator, with support from Cisco[5]. These clusters included a custom PDF guide, all the software required to operate the electronics kits via Python, and all the software required to operate an Ethereum network on each of the clusters, including PlaTIBART being installed on each cluster's first Raspberry Pi device.

6.8.2. In-Depth Guided Walkthrough

The benefit to the design of PlaTIBART is that the only difference between creating a Dockerized container containing all the required code to allow students to easily start learning Blockchain and configuring IoT workshop clusters is slight modification of the input model file and changing the files distributed to each client. Figure 14 shows the model input we used for creating the Docker image with a single client and a single miner, both on the same host, using localhost 127.0.0.1, but the network manager handles giving them ports in different ranges.

[5] https://oh-iot.com/

```
{" configurationName ":" test network b001 ",
 " configurationVersion ":"1",
 " chainId ": 15 ,
 " difficulty ": 100000 ,
 " gasLimit ": 200000000000000000 ,
 " balance ": 40000000000000000000000000 ,
 " genesisBlockOutFil e ":" genesis - data . json ",
 " clients ": {
          " startPort ": 9000 ,
           " client ":{ " count ": 1,  " hosts ": [ "127.0.0.1" ] }
          " miner ": { " count ": 1,  " hosts ": [ "127.0.0.1" ] }
}}
```

Figure 14 Blockchain assignment sample model json file

Now to support deploying to actual individual ITBA clusters the only requirement is that each system already have the host machine's public SSH key, have geth installed, and accept SSH connections.

```
{" configurationName ":" test network c001 ",
 " configurationVersion ":"1",
 " chainId ": 15 ,
 " difficulty ": 100000 ,
 " gasLimit ": 200000000000000000 ,
 " balance ": 40000000000000000000000000 ,
 " genesisBlockOutFil e ":" genesis - data . json ",
 " clients ": {
          " startPort ": 9000 ,
          " client ":{ " count ": 4,  " hosts ": [
                              "10.0.1.1","10.0.1.2","10.0.1.3","10.0.1.4" ] }
          " miner ": { " count ": 1,  " hosts ": [ "127.0.0.1" ] }
}}
```

Figure 15 IoT/Blockchain-Cluster configuration sample model json file

Figure 15 shows that the only change needed is adjusting the 'client' section of the json. Increasing the number of clients and changing what IPs the clients will be installed on. Both scenarios were completed via the exact same command line instructions as discussed in section 6.6.1 and 6.6.2, the only difference being what files were distributed to each client.

6.8.3. System Output and Analysis

Both examples discussed in this section proved to be viable and were successful in creating ITBA component educational assessments. Both Docker images of Blockchain assignments for university courses, and IoT and Blockchain workshop preparation and instruction were successful. However, neither of these were studied in-depth for formal verification, but instead were used as a means of rapidly creating repeatable Blockchain test networks in the educational scenarios. Therefore, these examples show more the adaptability of the PlaTIBART design when crafting future ITBA assignments. Future work will need to verify the efficacy of this approach.

6.9. Research Directions in **Testing at Scale of IoT & Distributed Systems**

Testing at scale of distributed systems is an ongoing research focus that will simultaneously have many different approaches. Formal validation of Blockchain related contract languages, systems, and tools, including PlaTIBART, is one area that will see continued focus. Additionally, verification of the efficacy of using PlaTIBART in an educational environment to teach ITBAs needs to be proven; this is a research area which we are currently pursuing. Expanding the network topology supported by PlaTIBART is future work that needs to be addressed, possibly with the integration with network simulation or management software. Work needs to be done on including private test Blockchain networks into both Unit and Integration Testing frameworks, and currently there don't appear to be tools, at least for Ethereum, for integration testing outside of Solidity IDE.

Key Terminology & Definitions

IoT Blockchain Applications (ITBAs) - Blockchain Applications that run on an IoT system where Blockchain is leveraged for a wide range of potential uses ranging from distributed logging up to integration into the command and control decision making process of the IoT device. Both Blockchain and IoT use case scenarios can change drastically when they are used together in a system. Therefore, we use this term to describe the

added complexity of Blockchain applications that interact with the physical world through IoT devices.

Testing-at-Scale - Testing distributed systems incurs a much heavier cost, both in complexity and required resources, when attempting to test a fully integrated distributed system versus traditional systems. This is because fully testing a distributed system requires a large amount of potentially heterogeneous devices running on potentially multiple platforms and/or architectures. This not only requires a system and methodology of testing that can be run easily with each new variation of the overall system, be it hardware or software, but also allows for consistent benchmarking, profiling, and analysis of performance, reliability, and other metrics.

Authors Bio:

Mr. Michael A. Walker is a Graduate Research Assistant pursuing his PhD in Computer Science at Vanderbilt University, Nashville, TN, USA. Currently he is an Instructor for the Computer Science & Information Systems department at Youngstown State University. He previously received his Masters in Science in Computer Science from Vanderbilt University [2011-2013], and obtained his Bachelors of Science in Computer Science from Youngstown State University [2006-2011].

Mr. Walker's research interests include Distributed Systems, Learning-at-Scale, Privacy, Security, and Software Design Patterns. He has published more than eleven research papers in various conferences, workshops and international journals of repute, including IEEE and ACM. He has been involved with ten Massive Open Online Courses, acting as a Teaching Staff for three and an Instructor for four courses. He is a present and past board member of several non-profits directed toward outreach and education of Science, Technology, Engineering, and Mathematics, with a concentration on computer literacy and scientific understanding, specifically focused on benefiting young girls from disadvantaged backgrounds. Additionally, he has given several conference presentations on the subject of bridging the academic and industry divide for non-traditional students.

Affiliation/Address:

E-mail: michael.a.walker.1@vanderbilt.edu
Affiliation
Institute for Software Integrated Systems
***Vanderbilt University*, Nashville, TN, USA**

Dr. Douglas C. Schmidt is the Cornelius Vanderbilt Professor of Computer Science, Associate Provost for Research Development and Technologies, Co-Chair of the Data Sciences Institute, and a Senior Researcher at the Institute for Software Integrated Systems, all at Vanderbilt University. His research covers a range of software-related topics, including patterns, optimization techniques, and empirical analyses of middleware frameworks for distributed real-time embedded systems and mobile cloud computing applications.

Dr. Schmidt has published 12 books and more than 600 technical papers covering a range of software-related topics, including patterns, optimization techniques, and empirical analyses of frameworks and model-driven engineering tools that facilitate the development of mission-critical middleware and mobile cloud computing applications running over wireless/wired networks and embedded system interconnects. For the past three decades, Dr. Schmidt has led the development of ACE and TAO, which are open-source middleware frameworks that constitute some of the most successful examples of software R&D ever transitioned from research to industry.

Dr. Schmidt received B.A. and M.A. degrees in Sociology from the College of William and Mary in Williamsburg, Virginia, and an M.S. and a Ph.D. in Computer Science from the University of California, Irvine in 1984, 1986, 1990, and 1994, respectively.

Affiliation/Address:

E-mail: d.schmidt@vanderbilt.edu
Affiliation
Institute for Software Integrated Systems
***Vanderbilt University*, Nashville, TN, USA**

Dr. Abhishek Dubey is an Assistant Professor of Electrical Engineering and Computer Science at Vanderbilt University, Senior Research Scientist at the Institute for Software-Integrated Systems and co-lead for the Vanderbilt Initiative for Smart Cities Operations and Research (VISOR). His research interests include model-driven and data-driven techniques for dynamic and resilient human cyber physical systems. He directs the Smart computing laboratory (scope.isis.vanderbilt.edu) at the university. The lab conducts research at the intersection of Distributed Systems, Big Data, and Cyber Physical System, especially in the domain of transportation and electrical networks. Abhishek completed his PhD in Electrical Engineering from Vanderbilt University in 2009. He received his M.S. in Electrical Engineering from Vanderbilt University in August 2005 and completed his undergraduate studies in Electrical Engineering from the Indian Institute of Technology, Banaras Hindu University, India in May 2001. He is a senior member of IEEE.

Affiliation/Address:

E-mail: abhishek.dubey@vanderbilt.edu
Affiliation
Institute for Software Integrated Systems
***Vanderbilt University*, Nashville, TN, USA**

REFERENCES

[1] Agrawal, Hiralal, Joseph Robert Horgan, Edward W Krauser, and Saul A London. "Incremental regression testing." In *Software Maintenance, 1993. CSM93, Proceedings., Conference on*, 348–357. IEEE, 1993.

[2] Banafa, Ahmed. *IoT and Blockchain Convergence: Benefits and Challenges - IEEE Internet of Things*. https://iot.ieee.org/newsletter/january2017/iot-and-blockchain-convergence-benefits-and-challenges. htm|. (Accessed on 08/31/2017), 2017.

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

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

[5] Bogner, Andreas, Mathieu Chanson, and Arne Meeuw. "A Decentralised Sharing App Running a Smart Contract on the Ethereum Blockchain." In Proceedings of *the 6th International Conference on the Internet of Things*, 177–178. IoT'16. Stuttgart, Germany: ACM, 2016. isbn: 978-1-4503-48140. doi:10.1145/2991561.2998465. http://doi.acm.org/10.1145/2991561.2998465.

[6] "Brooklyn Microgrid." 2017. http://brooklynmicrogrid.com/.

[7] Buccafurri, Francesco, Gianluca Lax, Serena Nicolazzo, and Antonino Nocera. "Overcoming Limits of Blockchain for IoT Applications." In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 26:1–26:6. ARES '17. Reggio Calabria, Italy: ACM, 2017. isbn: 978-1-45035257-4. doi:10.1145/3098954.3098983. http://doi.acm.org/10.1145/3098954.3098983.

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

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

[10] Company, Sterling Ranch Development. "The Nature of Sterling Ranch." 2017. http://sterlingranchcolorado.com/about/.

[11] Delmolino, Kevin, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. "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*, 79–94. Springer, 2016.

[12] Dorri, Ali, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. “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*, 618–623. IEEE, 2017.

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

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

[15] *Ethereum (ETH) $381.84 (3.83%) | CoinMarketCap*. https://coinmarketcap.com/currencies/ethereum/. (Accessed on 08/30/2017), August 2017.

[16] Gubbi, Jayavardhana, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. “Internet of Things (IoT): A vision, architectural elements, and future directions.” *Future generation computer systems* 29, no. 7 (2013): 1645–1660.

[17] Hintjens, Pieter. “ZeroMQ: The Guide.” *URL http://zeromq. org*, 2010.

[18] *Interfaces | Ethereum Frontier Guide*. https://ethereum.gitbooks.io/front ier-guide/content/interfaces.html. (Accessed on 08/30/2017), 2017.

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

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

[21] Lee, Edward A, Stephen Neuendorffer, and Michael J Wirthlin. “Actor-oriented design of embedded hardware and software systems.” *Journal of circuits, systems, and computers* 12, no. 03 (2003): 231–260.

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

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

[24] Melton, R. B. *Gridwise transactive energy framework*. Technical report. Pacific Northwest National Laboratory, 2013.

[25] Mirkovic, Jelena, and Terry Benzel. “Teaching cybersecurity with DeterLab.” *IEEE Security & Privacy* 10, no. 1 (2012): 73–76.

[26] Ouaddah, Aafaf, Anas Abou Elkalam, and Abdellah Ait Ouahman. “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*, 523–533. Springer, 2017.

[27] Rothermel, Gregg, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. “Prioritizing test cases for regression testing.” *IEEE Transactions on software engineering* 27, no. 10 (2001): 929–948.

[28] Siaterlis, Christos, Andres Perez Garcia, and Béla Genge. “On the use of Emulab testbeds for scientifically rigorous experiments.” *IEEE Communications Surveys & Tutorials* 15, no. 2 (2013): 929–942.

[29] Simić, Miloš, Goran Sladić, and Branko Milosavljević. “A Case Study IoT and Blockchain powered Healthcare,” June 2017.

[30] *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. (Accessed on 09/06/2017).

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

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

[33] Varda, Kenton. *Cap'n Proto*, 2015.

[34] Walker, Michael A., Abhishek Dubey, Aron Laszka, and Douglas C. Schmidt. "Platibart: a platform for transactive iot blockchain applications with repeatable testing." In Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things, pp. 17-22. ACM, 2017.

[35] Walker, Michael, Douglas C. Schmidt, and Jules White. "An elastic platform for large-scale assessment of software assignments for MOOCs (EPLASAM)." In User-centered design strategies for massive open online courses (MOOCs), pp. 187-206. IGI Global, 2016.

[36] Zhang, Fan, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. "Town Crier: An Authenticated Data Feed for Smart Contracts." In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 270–282. CCS '16. Vienna, Austria: ACM, 2016. isbn: 978-1-45034139-4. doi:10.1145/2976749.2978326. http://doi.acm.org/10.1145/2976749.2978326.