Proteus: A DLT-Agnostic Emulation and Analysis Framework

Russell Van Dam
Sandia National Laboratories

Thien-Nam Dinh
Sandia National Laboratories

Christopher Cordi
Sandia National Laboratories

Gregory Jacobus
Sandia National Laboratories

Nicholas Pattengale
Sandia National Laboratories

Steven Elliott
Sandia National Laboratories

Abstract

This paper presents Proteus, a framework for conducting rapid, emulation-based analysis of distributed ledger technologies (DLTs) using FIREWHEEL, an orchestration tool that assists a user in building, controlling, observing, and analyzing realistic experiments of distributed systems. Proteus is designed to support any DLT that has some form of a "transaction" and which operates on a peer-to-peer network layer. Proteus provides a framework for an investigator to set up a network of nodes, execute rich agent-driven behaviors, and extract run-time observations. Proteus relies on common features of DLTs to define agent-driven scenarios in a DLT-agnostic way allowing for those scenarios to be executed against different DLTs. We demonstrate the utility of using Proteus by executing a 51% attack on an emulated Ethereum network containing 2000 nodes.

1 Introduction

Since the creation of Bitcoin in 2008 [32], distributed ledgers, including blockchains, have exploded in popularity and span many real-world applications. Within each domain, there are numerous competing systems, each with a unique set of features including computation [45], privacy [40], and performance and scaling [38]. After Bitcoin's release, much of the blockchain innovation focused specifically on enhancing permissionless cryptocurrency ecosystems, of which there are now over 2,100 unique currencies [1]. However, aspects of this technology have been applied to more traditional financial systems by way of permissioned ledgers [10] and federated byzantine agreement models [30]. Additionally, new distributed ledger systems were developed that no longer use a traditional "blockchain" but rather use directed acyclic graphs (DAGs) [5] or a Hashgraph [4, 8]. We refer to all of these systems collectively as Distributed Ledger Technologies (DLTs) and consider all such systems to be within our domain of interest.

The rapid growth of new DLTs has far outpaced the creation of tools by which these systems can be analyzed [39]. Blockchain software developers have indicated a need for more effective tools for testing and security analysis [14]. For researchers, this dearth of testing capabilities also creates challenges for understanding DLTs at the system level. To address this gap, we propose leveraging recent advances in emulation capabilities. In such situations where mathematical simulations are insufficient and real-world experimentation is infeasible, emulation provides a realistic and safe research environment, allowing researchers the flexibility to test the effects of pathological or even malicious behaviors on real software.

Despite the rich diversity of DLTs, all of these technologies share a few core traits. First, they can be described as decentralized systems that communicate over the standard Internet Protocol. Second, they implement some concept of an (often peer-to-peer) overlay network. Finally, participating actors interact through some form of a transaction or a transaction-like event. These commonalities introduce a compelling opportunity for the creation of a DLT-agnostic testing platform.

We developed Proteus as an extension to FIREWHEEL, an experimentation platform, to take advantage of DLT commonalities and enable efficient testing and analysis of DLT software. We demonstrate that by using Proteus, a researcher can understand quickly the system-level impacts of a new mining approach or even assess the security implications of a given attack. Proteus is not intended to encompass the minutia of each DLT; by using a general framework, we can examine behavioral differences between DLT protocols, execute and observe network-level attacks, and discover the security implications of software vulnerabilities. Our primary contributions include:

- An orchestration framework that minimizes the time a user must spend on analyzing a new DLT.
- A method of programmatically controlling emulated DLT user behavior so that complex experiments can be constructed.
- An analysis of a 51% attack executed against an emulated Ethereum network containing 2000 nodes.

2 Background and Related Work

This section details the necessary background on the FIREWHEEL experiment orchestration tool, upon which Proteus relies, and outlines previous work on distributed ledger technology experimentation and analysis.

2.1 FIREWHEEL

FIREWHEEL [20] is an experiment orchestration tool that assists a user in building, controlling, observing, and analyzing repeatable experiments of distributed network systems at any scale. FIREWHEEL enables users to define models of network topologies and any time-scheduled actions they wish to performed within the experiment. At run-time, FIREWHEEL first represents a topology abstractly, using a graph data-structure, and then instantiates the topology across a cluster of network-connected servers and triggers the execution of scheduled actions at their appointed times. While many experiments are instantiated via emulation using KVM/QEMU [12, 27] and Open vSwitch [36], FIREWHEEL can utilize a variety of execution platforms including bare metal, emulation, simulation, or a combination of these approaches.

FIREWHEEL enables a researcher to 1) programmatically define and manipulate an experimental topology, 2) deploy the topology across a compute cluster, 3) manage the execution of in-experiment events, 4) centrally collect, analyze, and display experimental data, and 5) maintain an experimental description to reliably repeat an experiment. FIREWHEEL also includes a growing library of model components that ease construction of common topology features such as Linux and Windows hosts, routers, Internet services, and Windows services.

FIREWHEEL runs on commodity compute and networking hardware and a FIREWHEEL cluster can vary in size from a single laptop to hundreds of high-performance compute nodes.

Model Components Model components are the building blocks of FIREWHEEL experiments. Users build experiment models by creating and combining model components that define the topologies, attributes, configurations, and scheduled actions for their experiments. Essentially, a model component is a collection of files required to accomplish a specific objective with some additional metadata that identifies it to FIREWHEEL. Model components can contain Python code, which can create high-level abstractions of the experiment nodes and edges, virtual machine (VM) resources that are used to manipulate the VMs, and VM images. Model components can depend on the functions of other model components, making it easy to build complex experiments quickly. The model components developed for Proteus are described in Section 3.

Control Once an experiment has been defined through a set of model components, FIREWHEEL's *Control* system will translate the set of model components into an internal in-memory graph representation of the experiment. This graph, backed by NetworkX [25], contains all the information about each VM (the graph vertices) and the network connections amongst them (the graph edges) that were collectively specified by the model components. Once constructed, the experiment graph is used to instantiate a topology via a configurable launching mechanism. Proteus currently uses the default launching mechanism for FIREWHEEL (QEMU, KVM, and Open vSwitch) to begin an experiment.

VM Resource Handler Each node in the experiment may also have a schedule, in which each schedule entry outlines the command or executable to run, which files or resources are required to accomplish the action, and when the action is to occur. These schedule entries are defined by the model components and are propagated into the experiment graph. Once an experiment has launched, FIREWHEEL's *VM Resource Handler* monitors and manages the execution of VM schedule entries. Actions can be identified as either *pre* or *post* experiment start time. Pre-start time actions are used to configure each VM and prepare it for the experiment. These actions typically involve installing or configuring software. Once all VMs are configured, the Network Time Protocol is used to sync the time across the cluster and ensure a consistent experiment state. Next, the post-start time actions occur in accordance with their schedule.

FIREWHEEL redirects `stdout` and `stderr` from each VM resource into log files, which are ingested into Elasticsearch [2]. Then, using Kibana [6], a front-end for Elasticsearch, users can easily analyze experimental results, as shown in Section 5.2.

Networking FIREWHEEL supports emulating switches and routers: switches by bridging the interfaces of experiment VMs and routers by launching a router-based VM (typically VyOS). Additionally, FIREWHEEL has the ability to alter parameters on network links including bandwidth and latency[1]. Once an experiment is launched, FIREWHEEL's only interaction with the experiment networking is to ensure connectivity and adhere to the link-based properties set by the user. The remaining network interactions (OSI layers 2-7) are contained within the experiment and depend on the VMs and their installed software. When running in cluster mode, FIREWHEEL isolates experiment traffic between nodes by using either GRE tunnels or VLANs.

[1]FIREWHEEL uses the Traffic Control (tc) [9] package to manipulate traffic. Therefore, any parameters available through that package are usable within an experiment.

2.2 Related Work

When conducting analyses on blockchain systems, existing testing methodologies can be categorized into simulations, emulations, and physical tests. Each approach has trade-offs, involving factors such as experiment fidelity, computational cost, and reproducibility.

Simulators enable a researcher to imitate the processes of the software being tested. Typically, researchers can input a series of parameters that informs the system about timing information. These systems then provide time-based event logs that can be analyzed. Some simulators, such as those built in [22] and [23] are cryptocurrency-specific while others, including simbit [15], VIBES [42], and those presented in [24, 43, 46], enable simulating a diverse, but generalized set of blockchain parameters. Except simbit and the simulator developed by Gervais et al. [22], these papers do not consider network-level effects in their simulations and are unable to provide insight into the peer-to-peer network structure of the protocols. Furthermore, none of the aforementioned simulators run the actual DLT software implementations, which makes analysis of specific application nuances impossible.

Shadow [31], a discrete-event network simulator both accounts for network artifacts and permits running the `bitcoind` application to enable analyzing implementation artifacts. While the initial blockchain-related Shadow plugin is Bitcoin, Shadow can likely simulate many blockchain programs. However, there are some limitations to this approach. First, Shadow does not support multi-processing, and it requires that any nonblocking calls poll I/O events. This limitation could require source-code modifications, which reduces fidelity and increases the implementation overhead for each new DLT. Second, Shadow does not provide an interface in which the network can be modified dynamically. For example, it would not be possible to explore how BGP routing decisions might impact the security of Bitcoin's peer-to-peer network.

Emulation, which is the approach used by Proteus, enables running unmodified software, typically in a virtual machine (VM). Geier et al. [21] uses SherlockFog as a mechanism to deploy an Ethereum network. This work does not include a framework for easily adapting an experiment to work with other DLTs. Additionally, the topology is written in the custom SherlockFog language rather than Python. Chen et al. [16] demonstrates that containers can be used as a lightweight alternative to full VMs and effectively provide higher fidelity blockchain experiments with minimal extra overhead. However, this approach lacks the ability to modify the underlying network topology.

Physical testbeds are the most resource intensive and least reproducible option for analyzing DLT environments. Bitcoin-NG [18] and TrustChain [35], which propose new DLTs, and Pongnumkul et al. [37], which administers a performance analysis between two DLTs, all conduct experiments via physical hardware. However, they do not use an orchestration framework, which decreases the repeatability of their results. Similarly, BLOCKBENCH [17], which provides a blockchain benchmarking framework, also does not use an orchestration framework, but provides the source code, which can be leveraged in replication. Currently, BLOCKBENCH does not have an easy way to automate running the test suite nor does it have the ability to manipulate and analyze network performance. Shbair et al. [41] developed a framework for analyzing private and permissioned blockchains using the Grid'5000 system. This experiment is more reproducible than other works, but is limited to the Grid'5000 platform. Their blockchain orchestration tool is tailored to provide a specific Ethereum network experiment rather than the generalizable approach taken by Proteus. In addition, Grid'5000 is a geographically distributed testbed, which may impact the network characteristics of the experiment.

Proteus builds on this body of previous work by providing a novel balance of developing high-fidelity and reproducible blockchain experiments that can be conducted on commodity hardware. Furthermore, we demonstrate that the architecture of Proteus enables dynamic DLT experimentation.

3 Proteus Design

Proteus is designed to be a modular framework, with experiments constructed by combining components at different layers to emulate a diverse set of possible experiments. A DLT supported by Proteus is abstracted into components, which enables the same experiment scenarios to be reused with any supported DLT. By making the initial implementation of new DLTs in Proteus an easy process, simple scenarios with new DLTs can be quickly emulated to conduct general purpose analysis. Proteus' design supports use cases including comparative analysis of different DLTs, assessing the security and privacy of a DLT against network attacks, evaluating the scaling properties of DLTs, and analyzing impacts of DLT software improvements.

Proteus adopts an Agent-based modeling paradigm [13, 29] to emulate realistic and diverse network behaviors at scale. Agents, represented in the FIREWHEEL graph as individual vertices, are each assigned a behavior that will be executed concurrently during the experiment. Proteus experiments start with each VM completing a series of synchronized setup phases to instantiate the DLT and complete peering. Each agent then begins autonomously executing its assigned behavior, which likely includes taking dynamic actions based on the evolving state of the network. This approach enables the ability to compose complex scenarios from simple components in a way that intuitively models the logic of real-world actors.

3.1 Proteus Agents

Proteus agents are defined by combining a component from each of five layers: *physical*, *peering*, *policy*, *adapter*,

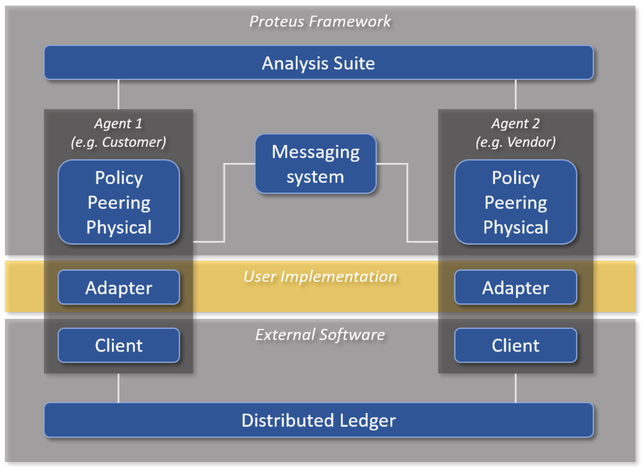


Figure 1: Proteus Architecture.

and *client*. Each component makes specific changes to the FIREWHEEL graph and assigns a relevant behavior to an agent. Except for the *physical* layer, which must be defined globally, each agent can be independently assigned one of many unique behaviors. Additionally, the Proteus framework provides infrastructure to enable communication and synchronization between agents, which we label as the *messaging system*. A high-level view of a Proteus experiment is illustrated in Figure 1.

While FIREWHEEL provides some underlying features used by Proteus, these layers were a fundamental enhancement to FIREWHEEL's existing capabilities by creating a new abstraction layer which can be leveraged for DLT-specific experimentation. Without the improvements made by Proteus, conducting large-scale dynamic DLT experiments via any emulation platform, including FIREWHEEL, would be time-consuming and require a high-degree of customization for each new experiment.

Physical Layer Proteus' physical layer component enables a user to define and customize the physical, data-link, and networking layers of their experiment. Thus, a user is able to mirror the physical infrastructure of a DLT network with an emulated replica, regardless of whether it is Internet-scale or a simple private network. Users create these topologies by programmatically adding switches and routers to the network graph and providing the IP addresses for each VM. Simple network topologies such as a "full mesh" or "star" can be used during initial testing and switched out later for more complicated topologies. These interchangeable topologies allow Proteus users to test latency issues, network bottlenecks, and model potential geographical bias of a DLT.

The transport layer of the networking stack is entirely dependent on the DLT software under test. For example, Bitcoin uses TCP as its transport protocol while Ethereum's Node Discovery Protocol uses UDP [3, 7]. Programmatic configuration of the transport layer is not supported within Proteus because it would require DLT-specific software modifications.

Peering Layer Most DLTs implement a network of peer-to-peer (P2P) connections overlaying the physical layer. The structure of this P2P network is an important factor in the robustness and security of the DLT's protocol. Proteus enables users to define static topologies, such as a "d-regular tree" [19], and dynamic topologies that adjust connections over time. Testing P2P topologies allows researchers to measure the effectiveness of transaction and block propagation as well as the resilience of the network against targeted attacks.

Policy Layer Policies are the core components enabling Agent-based modeling in Proteus. Each policy is a set of actions that aims to mimic typical behaviors in a DLT, including combinations of sending transactions, mining blocks, managing peering, and other well-defined actions. In this layer, each agent is assigned a specific policy, resulting in experiments that model complex scenarios such as a 51% attack or denial of service via transaction flooding. Experiments usually consist of a few agents behaving maliciously while all other agents are benign actors like miners, users, and observers.

Adapter Layer The adapter component is the interface between the policy and client components. Whereas the previously described model components implement abstract, DLT-agnostic behavior, the adapter is responsible for translating each policy's behaviors into concrete invocations of the underlying DLT software. We detail the interface an adapter implements in Section 4.2.

Client Layer Lastly, the client layer is responsible for scheduling the installation of the DLT and its various dependencies on each agent VM at experiment setup time. Proteus makes use of this component, although it depends on FIREWHEEL rather than Proteus itself. Although the client and adapter components are tightly coupled, there are two primary advantages to having them separate. First, the adapter could be used to test different versions of the same DLT software being installed by different client components. Second, since the client component does not rely on Proteus, it can be reused in other FIREWHEEL experiments.

3.2 Messaging System

The Proteus messaging system provides an in-experiment communication channel that enables agents to communicate amongst themselves. This system serves three primary purposes. First, it enables a mechanism for coordinated launch and teardown of the experiment at the application level, as described in Section 4.2. Second, agents can use the messaging system to pass "out-of-band" messages, such as wallet addresses or network information to another agent. Finally, the messaging system enables a global synchronization function allowing any number of agents to perform actions synchronously, which is necessary for modeling certain behaviors.

4 DLT Integration

The modular makeup of Proteus is intended to enable extensibility at various levels. Creating additional physical, peering, and policy layer behaviors is a straightforward process and any new implementation of these levels can be immediately re-purposed by other users. For example, the Proteus policy used in this paper to emulate a 51% Ethereum attack also works to emulate a 51% Bitcoin attack. However, one of the primary design goals of Proteus is to streamline the process of adding new DLTs. This is an intrinsically nuanced process, which we outline in the remainder of this section.

4.1 Client Creation

The process of creating a DLT client is analogous to setting up a node in the live DLT network. This includes dependency installation, fetching/building binaries, and node configuration. The exact steps are unique to each DLT, but the process is typically well-documented for users who want to join the live network. The challenges of translating this process into a Proteus client include 1) experiments are run in an isolated environment without Internet connection and 2) software installation is explicitly described in code. These challenges represent a necessary cost to ensure the reproducible and isolated nature of FIREWHEEL experiments.

To enable software installation in an offline environment, users need to save the target software and all of its dependencies. This can be performed easily by launching an Internet-connected VM and saving a cached copy of the necessary software. Many package managers have a "download" option that will assist in this process.

Once a user has captured the required software and dependencies, they must codify the process of installing these within their experiment environment. This burden is alleviated by leveraging FIREWHEEL's built-in functions to conduct common actions such as adding files to the VM, installing packages offline, and running executables. For reference, it takes less than 50 lines of Python code to develop a fully functioning Ethereum client.

4.2 Adapter Development

Proteus adapters were purposefully designed to have a simple application programming interface (API), supporting the rapid, initial data collection of an emulated, large-scale DLT experiment. However, they are often able to be tested only in the environment where they will be executed, since they have to communicate with a running instance of the DLT. This can dramatically increase the time to develop a new adapter, since their development usually requires multiple generations of code and creating the FIREWHEEL environment for each iteration adds notable overhead. To address this issue, we created an experiment configuration and GUI application where adapter updates can be pushed out to agents and the Proteus framework can be reset.

This tool, named *Blocky*, launches a single FIREWHEEL experiment with only a handful of VMs configured with the *client* component to initialize the DLT. The dashboard, displayed in Figure 2, connects to each VM and provides the user an interface to invoke individual API commands. Therefore, *Blocky* prevents having to repeatedly restart each experiment and allows developers to interactively and incrementally create a new adapter.

The following 11 methods of the adapter API supply the minimum amount of functionality to Proteus:

- *init_application* – Launch node/wallet software.
- *teardown* – Terminate node/wallet software.
- *get_applicationID* – Return DLT application identifier.
- *get_networkID* – Return IP address.
- *propose_block* – Append a new block to the ledger.
- *send_transaction* – Issue the provided transaction.
- *handle_custom* – Handle a user defined command.
- *get_balance* – Get the balance of the wallet (if applicable).
- *get_peers* – Get a list of the node's peers.
- *add_peer* – Connect the node with a given peer.
- *remove_peer* – Disconnect a peer from the node.

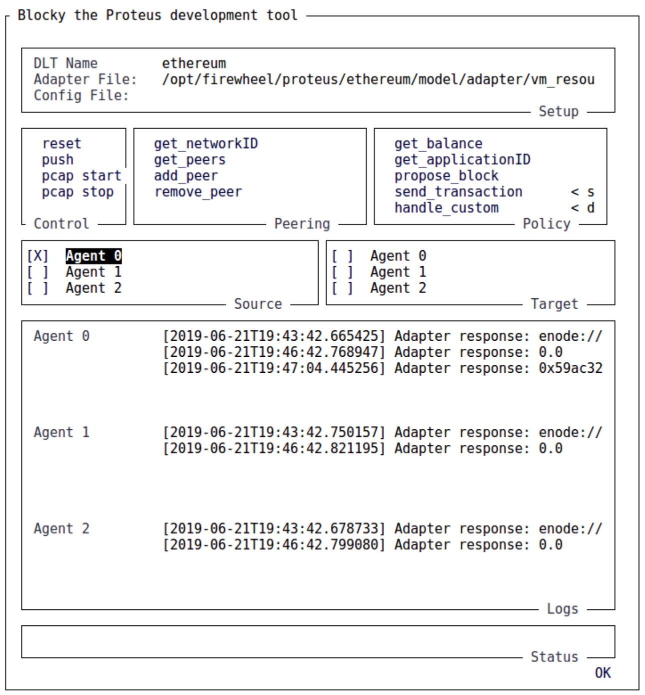
Using Blocky, a base Ethereum adapter can be created in a few hours by a developer comfortable with the go-ethereum API, though further customization may be needed for more complex experiments. The resulting adapter is roughly 350 lines of relatively simple Python code.

5 Evaluation

To evaluate the effectiveness of Proteus on a real-world scenario, we developed several policies that exercise Proteus' basic functionality. The most complex of these policies executes a 51% attack. In this section, we detail how we collect metrics from Proteus experiments and then provide details of running the 51% attack policy against an emulated Ethereum network.

5.1 Existing Analytics

When analyzing the effects of a new DLT, it is important to understand both system-level characteristics and DLT-specific information. Proteus uses the *Analytics* model component, which provides FIREWHEEL experiments with tools for monitoring and collecting data on each VM. Current capabili-



```
Blocky the Proteus development tool

  DLT Name         ethereum
  Adapter File:    /opt/firewheel/proteus/ethereum/model/adapter/vm_resou
  Config File:
                                                                  Setup

   reset          get_networkID            get_balance
   push           get_peers                get_applicationID
   pcap start     add_peer                 propose_block
   pcap stop      remove_peer              send_transaction      < s
                                           handle_custom         < d
  Control                        Peering                   Policy

 [X]  Agent 0                         [ ]  Agent 0
 [ ]  Agent 1                         [ ]  Agent 1
 [ ]  Agent 2                         [ ]  Agent 2
                             Source                            Target

  Agent 0          [2019-06-21T19:43:42.665425] Adapter response: enode://
                   [2019-06-21T19:46:42.768947] Adapter response: 0.0
                   [2019-06-21T19:47:04.445256] Adapter response: 0x59ac32


  Agent 1          [2019-06-21T19:43:42.750157] Adapter response: enode://
                   [2019-06-21T19:46:42.821195] Adapter response: 0.0


  Agent 2          [2019-06-21T19:43:42.678733] Adapter response: enode://
                   [2019-06-21T19:46:42.799080] Adapter response: 0.0

                                                                   Logs

                                                                 Status
                                                                     OK
```

Figure 2: Blocky: The Proteus adapter development tool.

ties include capturing network traffic; tracking network port changes; using `strace` on processes; capturing specified log files; tracking network and disk I/O; and monitoring CPU, memory, and disk utilization.

Proteus also collects DLT-specific information from each action being performed by the *Policy* that the *Agent* is executing. These logs include information about sent/received transactions, hash power, and wallet balances. This data is automatically ingested into Elasticsearch, enabling a researcher to visualize experiment results in real-time with Kibana.

5.2 Case Study: 51% Attack

The original Bitcoin [32] white paper noted that when a colluding set of malicious nodes controls a majority of the network mining power, they can subvert the security of the system and double-spend a transaction. This attack is colloquially known as a 51% attack. With respect to Bitcoin, this attack is largely thought to be impractical [11, 28]; however, less popular cryptocurrencies including Ethereum Classic [33], Bitcoin Gold [26], and Verge [44] have all had successful 51% attacks carried out against them in which attackers double-spent millions of dollars.

To gain insight into how to quickly detect and prevent these attacks, we ran multiple experiments using Proteus to emulate a 51% attack against Ethereum. We divided Ethereum nodes into malicious and honest partitions, where the malicious partition has 60% of the total network hash power. Figure 3 shows a scaled version of the initial peer-to-peer topology, in

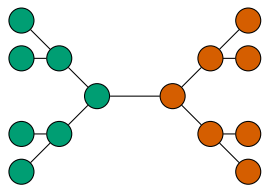


Figure 3: Example initial P2P topology of the 51% attack.

which nodes in each partition are peered amongst themselves in a regular tree and there is one cut edge between the partitions. The malicious partition policy enforces that malicious nodes exclusively peer with each other, except the malicious cut vertex (hereafter referred to as the boss), which ensures that it is peered to exactly one honest node.

The agents begin mining on a stabilized network, and all malicious agents send any mining rewards to the boss. Once the boss has sufficient funds, it broadcasts a transaction, which will be double-spent later, to the honest partition. Next, the boss removes its honest peer, disconnecting both partitions, and sends a double-spend transaction to the malicious partition, forking the chain. Because the malicious partition has more mining power, its blockchain will surpass the honest blockchain in height, at which point the boss publishes the malicious chain, completing the 51% attack.

Several network metrics may indicate whether this attack has occurred, including time to generate a new block, the network hash power, and blockchain reorganization depth. In this experiment, neither partition adds new hash power to the network. Therefore, when the malicious partition stops mining on the honest blockchain, the network hash power will drop and the time between new blocks will increase. Figure 4 shows the Ethereum network's hash power during the emulated 51% attack. During this experiment, the hash power decreases from approximately 308,000 hashes per second when the attack begins (around 22:28) to about 99,000 hashes per second when the attack is in full force (around 22:38)[2]. Figure 5, which illustrates block times, almost mirrors Figure 4, likely because these metrics are closely related. As expected, once the attack begins, the block time increases because the malicious agents started mining on a separate fork and the mining difficulty had not yet been adjusted to the decreased hash power of the network.

Blockchain reorganizations, known as reorgs, denote when a client identifies a chain that is longer than the one that the client previously thought was the longest. The client will then switch to the longest chain. Previously confirmed blocks that are now not on the longest chain become orphans. The number of orphaned blocks is referred to as the reorg depth. When the network is behaving honestly, these reorgs are small

[2]To help mitigate resource contention amongst experiment VMs, we artificially throttle mining new blocks by randomizing sleep/mine intervals. This causes a lower hash rate than would normally be expected.

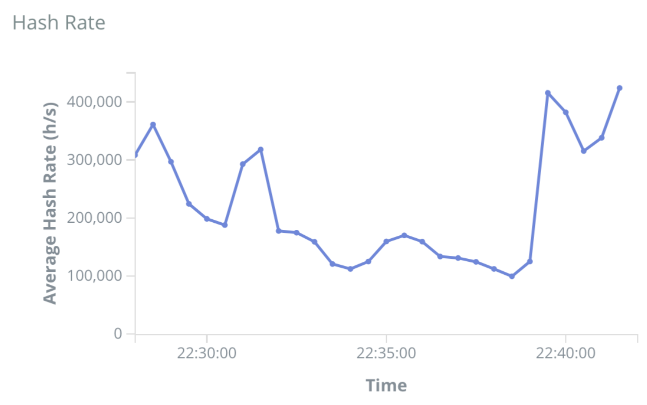


Figure 4: Ethereum hash power during the 51% attack.

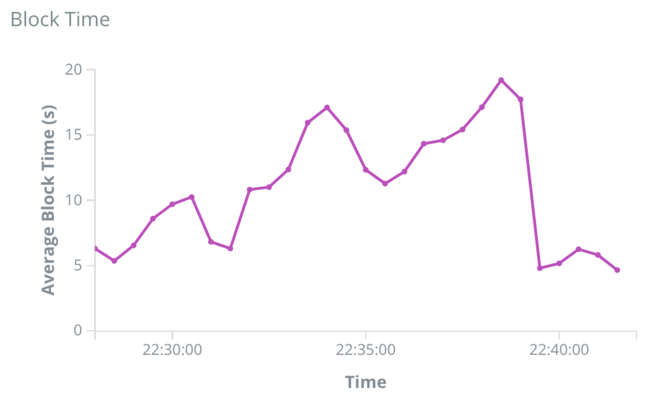


Figure 5: Ethereum block times during the 51% attack.

and tend to stay under a depth of 3. However, larger reorgs likely indicate malicious activity [34]. Figure 6 shows the reorganization depth throughout the 51% attack. When the attacker publishes the malicious chain (around 22:39), there is a large spike in the reorganization depth, as the honest agent adopts the malicious chain.

This example shows that large drops in hash power and spikes in reorg depth could indicate an increased risk of an impending 51% attack. However, we note that a drop in hash power would likely not occur if an attacker adds new resources that are not seen by the honest nodes rather than using computational resources that are already part of the network.

These experiments were conducted on a cluster of 20 FIREWHEEL nodes, each with dual socket Intel® Xeon® E5-v4 2.10GHz CPUs (16 cores per socket and HyperThreading enabled) and 512GB Memory. Each node has local solid-state drives and they are networked together with 100 Gigabit Ethernet. Our experiments included 2000 Ubuntu 16.04 Server VMs running go-ethereum-1.8.23. It took approximately 10 minutes from experiment launch until all the VMs were started, configured, and had their go-ethereum daemons

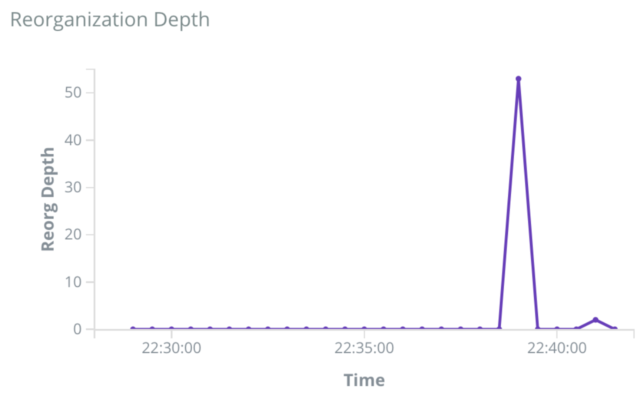


Figure 6: Ethereum reorg depth during the 51% attack.

peered. The malicious Proteus policy, which orchestrates the 51% attack, is approximately 450 lines of Python code.

6 Conclusion and Future Work

In this paper we presented Proteus, a framework for conducting rapid, emulated analysis of DLTs. First, we showed how Proteus uses agent-driven behavior to model complex scenarios in a DLT-agnostic way. Next, we described how users can quickly add a new DLT client and adapter. Lastly, we evaluated this framework by demonstrating a 51% attack on a private Ethereum network of 2000 nodes. Our results validate Proteus' utility in assessing the security guarantees of DLTs.

Proteus was designed to apply a single policy to multiple DLTs, which is an area of current research. Future testing may enable identification of cross-DLT analytics to detect common attacks on these platforms. We are also interested in developing adapters for non-cryptocurrency DLTs and assessing their compatibility with our framework. Lastly, we hope to incorporate real-world P2P topologies in Proteus and evaluate how they might impact the indicators of a 51% attack. For example, the attack outlined in this paper uses client-level peering to conduct the attack but a more realistic topology would include routing-level decisions that could impact the effects of this attack.

Availability and Funding

FIREWHEEL was developed at Sandia National Laboratories, which is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. There is a non-exclusive license for use of this work by or on behalf of the U.S. Government. We will be submitting FIREWHEEL and Proteus through the appropriate review

process so that these tools may become available to the wider community in the future.

References

[1] Cryptocurrency market capitalizations. https://coinmarketcap.com/.

[2] Elasticsearch: RESTful, Distributed Search & Analytics. https://www.elastic.co/products/elasticsearch.

[3] Ethereum Discovery Overview. https://github.com/ethereum/devp2p/wiki/Discovery-Overview.

[4] Hedera Hashgraph. http://www.hedera.com.

[5] IOTA. https://www.iota.org/.

[6] Kibana: Explore, Visualize, Discover Data. https://www.elastic.co/products/kibana.

[7] Network: Bitcoin Wiki. https://en.bitcoin.it/wiki/Network.

[8] Swirlds. https://www.swirlds.com/.

[9] Traffic control (tc). https://linux.die.net/man/8/tc.

[10] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the 13th EuroSys Conf.*, page 30. ACM, 2018.

[11] Osato Avan-Nomayo. Bitcoin 51% Attack is Unrealistic, New Study Concludes. https://bitcoinist.com/bitcoin-51-percent-attack-study/.

[12] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41. USENIX Association, 2005.

[13] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. 99:7280–7287, 2002.

[14] Amiangshu Bosu, Anindya Iqbal, Rifat Shahriyar, and Partha Chakroborty. Understanding the Motivations, Challenges and Needs of Blockchain Software Developers: A Survey. 2018.

[15] Sean Bowe. Simbit - Javascript P2P Network Simulator. https://github.com/ebfull/simbit.

[16] Chen Chen, Zhuyun Qi, Yirui Liu, and Kai Lei. Using Virtualization for Blockchain Testing. In Meikang Qiu, editor, *Smart Computing and Communication*, Lecture Notes in Computer Science, pages 289–299. Springer, 2018.

[17] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1085–1100. ACM, 2017.

[18] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. pages 45–59, 2016.

[19] Giulia Fanti and Pramod Viswanath. Deanonymization in the bitcoin p2p network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1364–1373. Curran Associates, Inc., 2017.

[20] Kasimir Georg Gabert, Adam Vail, Tan Q. Thai, Ian Burns, Michael J. McDonald, Steven Elliott, John Vivian Montoya, Jenna Marie Kallaher, and Stephen T. Jones. FIREWHEEL - a Platform for Cyber Analysis.

[21] Maximiliano Geier, Claudio J. Tessone, Marco Vanotti, Silvio Vileriño, David González Márquez, and Esteban Mocskos. Using Network Emulation to Study Blockchain Distributed Systems: The Ethereum Case. In *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 51–58. IEEE, 2019.

[22] Arthur Gervais, Ghassan Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the Security and Performance of Proof of Work Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference*, CCS '16, pages 3–16. ACM, 2016.

[23] Johannes Göbel, H. Paul Keeler, Anthony Krzesinski, and Peter Taylor. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. 104:23–41, 2016.

[24] Sneha Goswami. Scalability Analysis of Blockchains Through Blockchain Simulation, 2017.

[25] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, 2008.

[26] Edward Iskra. Double Spend Attacks on Exchanges. https://forum.bitcoingold.org/t/double-spend-attacks-on-exchanges/1362.

[27] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. KVM: The Linux Virtual Machine Monitor. In *In Proceedings of the 2007 Ottawa Linux Symposium (OLS'-07*, 2007.

[28] Joshua A. Kroll, Ian Davey, and Edward W. Felten. The Economics of Bitcoin Mining , or Bitcoin in the Presence of Adversaries.

[29] C. M. Macal and M. J. North. Tutorial on agent-based modeling and simulation. In *Proceedings of the Winter Simulation Conference, 2005.*, pages 14 pp.–, 2005.

[30] David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, 2015.

[31] Andrew Miller and Rob Jansen. Shadow-Bitcoin: Scalable Simulation via Direct Execution of Multi-Threaded Applications. 2015.

[32] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf.

[33] Mark Nesbitt. Ethereum Classic (ETC) is currently being 51% attacked. https://blog.coinbase.com/ethereum-classic-etc-is-currently-being-51-attacked-33be13ce32de.

[34] Mark Nesbitt. Ethereum Classic (ETC) is currently being 51% attacked, January 2019. https://blog.coinbase.com/ethereum-classic-etc-is-currently-being-51-attacked-33be13ce32de.

[35] Pim Otte, Martijn de Vos, and Johan Pouwelse. TrustChain: A Sybil-resistant scalable blockchain. *Future Generation Computer Systems*, 2017.

[36] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130. USENIX Association, 2015.

[37] Suporn Pongnumkul, Chaiyaphum Siripanpornchana, and Suttipong Thajchayapong. Performance Analysis of Private Blockchain Platforms in Varying Workloads. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6, 2017.

[38] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[39] Simone Porru, Andrea Pinna, Michele Marchesi, and Roberto Tonelli. Blockchain-Oriented Software Engineering: Challenges and New Directions. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 169–171.

[40] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[41] Wazen M. Shbair, Mathis Steichen, Jérôme François, and Radu State. Blockchain orchestration and experimentation framework: A case study of KYC. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6, 2018.

[42] Lyubomir Stoykov. VIBES: Fast Blockchain Simulations for Large-scale Peer-to-Peer Networks, 2018.

[43] Bozhi Wang, Shiping Chen, Lina Yao, Bin Liu, Xiwei Xu, and Liming Zhu. A Simulation Approach for Studying Behavior and Quality of Blockchain Networks. In Shiping Chen, Harry Wang, and Liang-Jie Zhang, editors, *Blockchain – ICBC 2018*, Lecture Notes in Computer Science, pages 18–31. Springer, 2018.

[44] Josiah Wilmoth. Privacy Coin Verge Succumbs to 51% Attack [Again]. https://www.ccn.com/privacy-coin-verge-succumbs-to-51-attack-again.

[45] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[46] Rajitha Yasaweerasinghelage, Mark Staples, and Ingo Weber. Predicting Latency of Blockchain-Based Systems Using Architectural Modelling and Simulation. In *2017 IEEE ICSA*, pages 253–256, 2017.