

**From the holdings of the Department of Special Collections , Princeton University Library,
Princeton, NJ**

These documents are either in the public domain, or can only be used for educational and research purposes (Fair use) as per U.S. Copyright law (text below). By accessing this file, all users agree that their use falls in the public domain or within fair use as defined by the copyright law. We do not charge any permission or usage fees for the publication of images of material in our collections, including those provided by Princeton University Library, via our website, catalog, or directly from staff. For more information regarding copyright, credit, and citation please review the information on the Copyright, Credit and Citation Guidelines page of our website.

<https://library.princeton.edu/special-collections/policies/copyright-credit-and-citation-guidelines>. If you have any questions, including how to order higher-quality images from the collection, contact Special Collections Public Services via our Ask Us! Form. <https://library.princeton.edu/special-collections/ask-us>

Master's Theses, 2022-2026

Electrical and Computer Engineering

2024

Proof-of-Stake Ethereum Emulation Platform For Analyzing Network Level Consensus Layer Attacks

Parikh, Ronak

Princeton, NJ : Princeton University

<http://arks.princeton.edu/ark:/88435/dsp018910jx970>

U.S. Copyright law: The copyright law of the United States (Title 17, §108, United States Code) governs the making of photocopies or other reproductions of copyrighted material. Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be used for any purpose other than private study, scholarship, or research. If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of fair use that user may be liable for copyright infringement. This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve a violation of copyright law. <https://www.copyright.gov/title17/title17.pdf>

PROOF-OF-STAKE ETHEREUM EMULATION PLATFORM FOR ANALYZING
NETWORK LEVEL CONSENSUS LAYER ATTACKS

Ronak Parikh

A MASTER'S THESIS
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF MASTER OF SCIENCE IN ENGINEERING

RECOMMENDED FOR ACCEPTANCE BY
THE DEPARTMENT OF
ELECTRICAL AND COMPUTER ENGINEERING

Adviser: Professor Maria Apostolaki

May 2024



This work is licensed under a Creative Commons “Attribution-NonCommercial-ShareAlike 4.0 International” license. (<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.en>)

Abstract

Cryptocurrencies have gained popularity recently as a monetary system with trillions of United States dollars entrusted to them. Thus, ensuring their security from attackers is crucial. In the past, the largest cryptocurrencies by market capitalization used the well studied Proof-of-Work (PoW) as their consensus protocol, but many have critiqued its unsustainable energy consumption. On September 15, 2022 the second largest cryptocurrency, Ethereum, switched to a Proof-of-Stake (PoS) consensus protocol with enormous energy savings, but the complexity of Proof-of-Stake in combination with Ethereum’s features makes it difficult to study its security guarantees. Ethereum has since updated further to patch vulnerabilities and add even more features. In this fast paced evolution, tools to analyze Ethereum, in its full complexity, under attack are necessary.

We present an Ethereum emulation and controller platform that enables analysis of attacks at the network level on the latest industry standard clients. We use these tools to emulate network delay and partitioning attacks on Ethereum and successfully visualize their measured effects on consensus. Our results point to this platform being useful for further research with attacks at the protocol and application layers, particularly those that use the network as an attack vector.

Acknowledgements

I would like to thank Professor Maria Apostolaki for her direction on this thesis over the past year. You have been an instrumental part of my education since I began my journey at Princeton two years ago in your classroom. Your teaching and mentorship have been invaluable and your commitment to education and research is inspirational.

I also want to thank my parents and brother for their unwavering love and support. You have made me everything I am today and I will never forget all that you have done for me.

Contents

Abstract	3
Acknowledgements	4
1 Background	7
1.1 Cryptocurrencies	7
1.2 Bitcoin & Proof-of-Work Ethereum	8
1.2.1 Ethereum Virtual Machine & Smart Contracts	13
1.3 Proof-of-Stake Ethereum	15
1.3.1 LMD GHOST & Casper FFG	18
1.3.2 Slashing & Inactivity Leak	19
1.3.3 RANDAO	20
1.3.4 Attacks on Consensus	21
2 Introduction	23
2.1 Related Work: Ethereum Simulators	23
2.2 Related Work: Container Based Ethereum Emulation	24
3 Proposed Design	25
3.1 Objectives	26
3.2 Contributions	26
3.3 Emulation Over Simulation	27
3.4 SEED Internet Emulator	28
4 Implementation	30
4.1 Setup	30
4.2 Data Collection	32
4.3 Controller	32

4.4	Limitations	33
5	Results	35
5.1	Partition Attack	35
5.2	Network Delay Attack	39
6	Future Work	43
7	Conclusion	44

1 Background

Popularized following the 2008 financial crisis, a set of alternative digital monetary systems separate from government issued currencies and centralized financial institutions is gaining increasing adoption. These monetary systems are decentralized cryptocurrencies and their security is the focus of this thesis. In particular we look at Proof-of-Stake Ethereum, the second largest cryptocurrency by market capitalization as of March 2024 and the largest Proof-of-Stake cryptocurrency.

1.1 Cryptocurrencies

Cryptocurrencies are built on blockchain data structures, a linked list made of blocks which are referenced by their hash and contain an ordered list of transactions. Cryptocurrencies are permissionless and dynamic, relying on distributed consensus of the blockchain among members, known as nodes, rather than a centralized ledger to record transactions. The permissionless aspect means that anyone can join and engage in transactions with other nodes and the dynamic aspect that nodes can disconnect and rejoin at any time. These nodes connect to each other and propagate transactions in a network built on top of the internet, known as an overlay network. Their openness and lack of reliance on centralized institutions has attracted trillions of dollars in investment. This has placed a large target on them and as such they have a history of attacks, including attacks by state backed groups. Their attack surface includes their protocol layer, applications built on top of them, and the underlying infrastructure that supports them, such as the Internet.

Cryptocurrencies can use different consensus mechanisms in their protocol layer to reach consensus about transactions in the ledger, two of which are Proof-of-Work (PoW) and Proof-of-Stake (PoS). Ethereum is currently the most widely adopted PoS cryptocurrency with a market capitalization of over 400 billion United States dollars as of March 2024. Initially, much like the current largest cryptocurrency by market capitalization, Bitcoin,

Ethereum used PoW for consensus. PoW has garnered criticism for its unsustainable energy consumption and wasteful hardware arms race; it was estimated that Ethereum used to use 21 terawatt hours per year. On September 15, 2022, Ethereum switched to PoS for consensus in a protocol upgrade known as “The Merge” (code named Paris). This new consensus mechanism brought Ethereum’s energy consumption down by an estimated 99.95%. However, it also brought increased complexity to Ethereum’s protocol which was already more complicated than Bitcoin’s.

1.2 Bitcoin & Proof-of-Work Ethereum

Bitcoin is the largest cryptocurrency and still uses PoW today, whereas PoW Ethereum is now deprecated. Built on asymmetric cryptography, Bitcoin defines an electronic coin, its namesake currency, as a chain of transactions [Nak09]. Each coin can be sent to a recipient by adding a transaction which consists of a hash of the previous transaction in which the sender received the coin, and the public key of the recipient. In actuality a Bitcoin transaction can include multiple input transactions and recipients, and this model is known as unspent transaction output (UTXO). Every transaction is signed with the sender’s private key to prove it is really from that sender. Ethereum, whose coin is referred to as an Ether (ETH), uses an essentially equivalent account model for transactions. A public key has an associated balance which is simply all of the coins they have received less all the coins they have sent in all previous transactions, as well as an account nonce that increments with each signed transaction sent from that account.

For both, nodes each randomly connect to a few of their peers. Transactions are propagated to all other nodes in the network with a gossip protocol, where nodes check that they are valid else they are ignored. Transactions are put into an ordered list in an object called a block which is added to the end of previous blocks in a data structure known as a blockchain. Each block, in addition to a list of transactions, contains a header with the hash of one previous block, its parent, which establishes an ordering of the blocks. In each header

there is also a nonce value, threshold or difficulty value, and the Merkle root of a Merkle tree of the transactions in the associated block. The nonce value makes up the Proof-of-Work for the block. Significant computation is expended trying different values of the nonce and hashing (SHA-256) the header with that value to see if it is in the set of acceptable hashes in a process called mining. That set is determined by the difficulty value which is decided on by the protocol to maintain a constant rate of new blocks and corresponds to the number of leading zeros the header of the block's hash must contain. Due to the one way property of hash functions, it is computationally easy to validate that the nonce results in the correct hash. Each block is also propagated to all other nodes in the network with a gossip protocol where nodes check that their transactions and header is valid else they are discarded. When a node finds the correct nonce and propagates the valid block it is said to have proposed it. The Merkle root in the header allows some nodes to only download headers of blocks if they don't care about all of the transactions while still being able to prove some transactions were in a given block should they need to later.

It is possible for there to be multiple blocks with the same parent if a node was mining on an old block due to a network delay or a deliberate attack, so each node looks at all the blocks they know about and decides on a canonical chain, the longest chain of blocks, which is proportional to the most total computational energy invested. Only the blocks in this longest chain are considered and all other branches, or forks, from this chain are ignored. Using this rule nodes can reach a distributed consensus about blocks in the canonical chain, especially for older blocks which are more likely to be known by most of the network and have more computational weight built on top of them through their descendant blocks. Given that blocks are ordered in the blockchain and within each block transactions are also ordered, in the longest chain there is only one sequential order for all transactions. This is what all of the nodes are ultimately coming to a consensus on. Since there is this ordering, rather than checking every previous transaction when verifying a new transaction, it is more computationally efficient to maintain a state at each block which contains information about

unspent transactions in Bitcoin's UTXO model or account balances and nonces in Ethereum's account model.

Economics Mining new blocks necessarily comes at a computational cost, so honest nodes need an incentive to mine blocks which keeps the network secure. This incentive comes in the form of block rewards which creates new coins in the first transaction of each new block and transaction or gas fees paid by any node who wants their transaction to be included in a block. Block rewards have functionality as a mechanism for distributing coins initially and as a source of controlled inflation. Bitcoin distributes block rewards only to the blocks that end up in the canonical chain, while PoW Ethereum differed by also giving a smaller reward to uncle blocks, also called Ommers blocks. Uncle blocks are ones which were proposed at nearly the same time as the block that ended up on the canonical chain and are typically caused by network latency. PoW Ethereum incentivized these blocks to prevent the nodes who proposed them from being heavily incentivized to build off them instead of the canonical chain since they were more likely to happen with PoW Ethereum's much shorter average block time of 13 seconds compared to Bitcoin's 10 minutes. Without that incentive many opportunistic nodes that proposed an uncle block might each start working on their own fork, fracturing the otherwise honest majority. Additionally, Ethereum also includes a burning feature whereby a portion of transaction fees are permanently removed from circulation by the protocol. This is useful as a deflationary tool and to disincentivize collusion between miners and users.

Basic Attacks The longest chain rule with an agreed upon sequential ordering of transactions paired with the PoW prevents a node from using the same coin in two different transactions at the same level in that coin's chain of transactions within a blockchain, an attack known as double spending. So long as the majority of computational power is honest, an attempt at double spending would be deemed invalid and only the first time the coin was spent would be valid. PoW also enables cryptocurrencies to be permissionless with resis-

tance to Sybil attacks since ability to propose is proportional to computational power which honest nodes are assumed to have a majority of. This assumption of an honest majority underpins most of Bitcoin's and Ethereum's security. Attempts to fork the blockchain would also require having more computational power than the honest majority to be successful, otherwise they would simply be outpaced. Further, the Merkle root of the transactions is included in the header so that a node propagating the block cannot tamper with the transactions without changing the Merkle root and thus the hash of the block, which would require needing to redo the PoW.

The very newest blocks at the tip of the longest chain may not be part of the canonical chain in the future. This is because there may be uncle blocks that nodes decide to build on instead due to network delay which end up becoming longer or because an attacker has a string of good luck and is able to propose multiple blocks from an older block that ends up being the new longest chain. To prevent double spending from happening in these scenarios there is a notion of finality or confirmation of transactions. The probability of an attacker having such a string of good luck diminishes exponentially with the number of blocks they need to catch up to with the longest chain. So, a confirmation scheme that arises from this is to wait for n blocks to be mined after the block a transaction is in on the canonical chain for that transaction to be considered "confirmed". The value of n is decided based on the risk tolerance of the parties involved in the transaction.

Routing Attacks Information is sent between Bitcoin nodes over the Internet and that infrastructure can be used as an attack vector. [AZV17] details two types of routing attacks against Bitcoin. The first uses a BGP hijack to attract all connections between Bitcoin nodes to a malicious AS and then drops those connections to partition the network. The set of nodes can represent a large portion of the Bitcoin network by mining power, and methodology for finding which nodes can be partitioned is outlined. During the partition, each side may start building on their own fork and this can allow for double spending and is

a general availability attack on the network. This attack is feasible, effective and quick but visible and can be short lived. The second routing attack is a delay attack, which modifies messages sent between targeted nodes and its peers via a malicious AS to delay their requests for information about the latest blocks. The attack is feasible without needing to intercept all of a target node's connections due to Bitcoin's peer connection policies which have long timeouts if strategically kept alive, and due to the lack of encryption and integrity checks between nodes' communication. The attack is invisible and also an attack on availability.

Privacy Since blocks and transactions must be publicly broadcasted, the entire record of economic activity is open for anyone to view which could be a drawback if privacy is a consideration. This activity can only be associated with pseudonymous addresses but those addresses can be de-anonymized by correlating the IP address where a transaction begins to propagate from with the transaction being propagated or by correlating transactions with off chain activity. Adjustments to the gossip protocol such as adding random delay schemes have been implemented to address the former and other privacy focused cryptocurrencies take measures to hide transaction data.

Mining Pools Computational power for mining is measured as a hash rate, which is the number of double SHA-256 hash computations per second in the case of Bitcoin and PoW Ethereum. Solo miners with limited hash rate compared to that of the entire network find that the time between their successful block proposals, and thus income source, is variable and lengthy due to the probabilistic nature of mining. In order to make this more regular many solo miners join collectives, called mining pools, with other miners and split income based on hash rate contributed. This however has had a centralizing effect, concentrating hash rate to a few pools. On Bitcoin, the four largest pools have mined over 75% of blocks in the past year. This has garnered criticism and concerns about security should one of these large pools act maliciously.

1.2.1 Ethereum Virtual Machine & Smart Contracts

Cryptocurrencies have properties that are useful for applications beyond basic currency functionality. Some of their useful properties include permissionless distributed consensus on the order of transactions, atomicity where a transaction either fully executes or fails, and consensus on a state transition function between each block's state. Applications which can use these properties include lending and other advanced financial instruments as well as name registration, and voting. With cryptocurrency being an alternative to centralized banking, a logical next step is supporting more advanced monetary tools that are offered in traditional finance as well as new tools only possible with the unique properties of blockchains. This ecosystem of financial applications built on top of cryptocurrencies is referred to as decentralized finance (DeFi). Bitcoin has some limited scripting functionality but this is not Turing complete and cannot hold any state between transactions, restricting the type of applications that can be built on top of it. Ethereum was created to open these blockchain properties up for a variety of applications by acting as a platform.

Ethereum has a Turing complete programming language built into its protocol which allows arbitrary applications to be built on top of it [But14]. These applications take the form of smart contracts, special accounts not owned by anyone with immutable open source code and storage in addition to the balance that all accounts have. An account can interact with a smart contract by creating a transaction that sends a message to the contract calling on one of its functions, contracts don't continuously run independently. Upon receiving that message, the contract can run code, modify its storage (its state), and send more messages to other smart contracts or deploy more smart contracts, all of this happens within that one initial transaction. Each code instruction the contract executes or storage amount used incurs a transaction fee known as gas in order to compensate the miners and to protect against spam.

The contract code is in a low level bytecode language called Ethereum Virtual Machine (EVM) code. Execution works by incrementing a program counter and carrying out each

associated operation until the code terminates or encounters an error. Running code has access to ephemeral and persistent storage as well as some other important values such as parameters in the transaction that called it and the state of the blockchain and can return a value. This code is actually executed in each state update with each new block, so all nodes verifying a new block will run any code called by any transactions in the block. A hash of the final state after applying all transactions in the block to the previous state in order is included in Ethereum blocks for verification. Contract code is typically written in a high level language such as Solidity and compiled down to bytecode to be deployed in a contract.

On one hand, smart contracts add a new layer of complexity, an application layer, and consequently add to the attack surface. Now an attacker's goal may not be to disrupt the entire currency but just one small application and mistakes in a contract's code can be exploited independent of the cryptocurrency's protocol layer. On the other hand, smart contracts enable a plethora of decentralized applications, financially related or not. One DeFi example is a flash loan, which allows for unlimited borrowing with no collateral within one block. If the loan is not paid back within the block the transaction fails, making use of cryptocurrency's atomicity properties. One non-DeFi example is a name registration system similar to DNS where a smart contract holds a database of names and their owners. Countless applications are already built on Ethereum smart contracts and see real world use.

DAOs An important application of smart contracts is a decentralized autonomous organization (DAO). DAOs are decentralized versions of democratic collective organizations like charities, grants, corporations, or governments. In one type of DAO, backers pool currency (ETH) into the DAO smart contract and receive a token representing their deposit. The group is then able to conduct votes on proposals such as how those funds are spent and other decisions by sending transactions to the smart contract while possessing the DAO's token. The DAO smart contract automatically enacts on passing votes based on its publicly viewable immutable code. Since smart contracts can hold a balance, no trusted centralized

treasury is needed to manage the group’s funds. Members can also entrust their vote to delegates since many proposals are often technical. Delegates make voting more centralized and are a potential weakness, and many popular DAOs have most of their tokens concentrated to only a few people. Further criticisms of DAOs are that what becomes a proposal is subject to social influence and voters can have low turnout or accept bribes. A critique of many smart contract applications is that duplicates could be deployed fracturing the userbase, and bugs in a contract’s code can be exploited heavily. Both DAOs and other smart contract applications have had a history of attacks in recent years. In one case, attackers used a flash loan to gain control of the majority of a DAO’s tokens, passed a vote giving them all of its funds, and repaid the loan, all in one block.

1.3 Proof-of-Stake Ethereum

Proof-of-Work necessarily derives security from expending energy and compute resources. Many view this as unsustainable and impractical if cryptocurrencies are meant to run a large portion of the world economy. At its heart, PoW relies on compute being a scarce resource that honest participants control the majority of. An alternative consensus mechanism, Proof-of-Stake, seeks to exchange this physical resource with a virtual one, the coin itself. On December 1, 2020, Ethereum’s creators launched a second blockchain, the Beacon chain, which implemented PoS consensus and on September 15, 2022, Ethereum switched to actually using that Beacon chain for consensus, in a protocol upgrade known as “The Merge”, with its original blockchain, also known as the Mainnet, becoming an execution layer working in parallel with the Beacon chain.

Instead of miners investing in computer hardware, PoS has validators that invest stake, the coin, into the protocol. For Ethereum, a validator must deposit at least 32 Ether (ETH) into a special smart contract, the Beacon deposit contract, and join an activation queue that limits the rate of new validators joining the network [ethb]. As validators they are responsible for checking the validity of new blocks and the transactions and state update within those

blocks, and submitting a vote for certain blocks, called an attestation. They also have a chance at proposing a new block themselves proportional to the amount of capital they have staked. If they are able to propose a new block that is accepted by the other validators, then they earn block rewards and transaction fees as with PoW, and attestations also earn a smaller reward, so that is their incentive to stake ETH. Should they act maliciously, by sending conflicting attestations or block proposals, all or part of their stake is slashed, or destroyed. Missing, late or incorrect attestations are also given a penalty which is much smaller than being slashed since that isn't always malicious but puts a strain on the security of consensus.

With PoS, blocks are proposed at exact fixed intervals called slots. In the case of Ethereum, slots are spaced 12 seconds apart and 32 slots make up an epoch. Every epoch, validators are chosen to be proposers for each slot weighted by their stake using RANDAO, an on-chain pseudorandom number generator. All active validators are assigned to at least one committee within a slot in the epoch by RANDAO where they act as attesters. During that slot they submit an attestation which is an LMD GHOST vote and Casper FFG vote weighted by their stake. Attestations enable fork choice as well as finality. These attestations are broadcast to the network like blocks and transactions, but before they are broadcasted, they are aggregated to save bandwidth. In each committee, some validators are randomly picked to be aggregators that collect all the attestations from their committee and combine them into a single attestation which they sign that includes a proof, a combination of all the signatures in the individual attestations. This signature combination is possible due to the properties of the type of signatures used on Ethereum, BLS signatures. BLS signatures allow signatures to be combined into an aggregate signature and checked by combining all of the associated public keys into an aggregate key. The aggregators are picked by revealing if their signature of the current slot number is modulo a protocol specified value.

Each beacon block, signed by the proposer, contains a header similar to PoW including the slot, parent block hash, state hash, and an ID corresponding to the proposer. In its body,

among other fields, are the randao reveal, list of slashings, list of attestations, deposits to the Beacon deposit contract, list of validators exiting the network, and an execution payload. The execution payload is an entire execution chain block wrapped inside the beacon block. This block is very similar to an Ethereum PoW block with its header containing among other fields, the parent execution block hash, timestamp, address of the proposer for transaction fees to be sent to, hash of the new state after applying the new transactions within the block, total gas or transaction fees used, and the Merkle root of the transactions in the body and of the withdrawals. The body contains the new transactions in the block as well as a list of withdrawals. The timestamp is used to keep beacon chain clients synchronized. Withdrawals are part of a very recent upgrade on April 12, 2023, that finally allowed stakers to withdraw their block rewards or exit staking and reclaim their deposit. Before that point when a staker joined with a 32 ETH deposit they weren't able to leave or even receive their interest from block rewards and transaction fees. These withdrawals are separate from transactions so that there are no transaction fees and are rate limited to 16 per block. Validators wanting to exit entirely also have to wait until a specific epoch where they are allowed to. These queues are in place to prevent a sudden surge of withdrawals that could destabilize Ethereum's security.

Clients For a node participating in PoS Ethereum as a validator, they must run 3 clients: an execution client, a validator client, and a beacon client. The beacon and validator clients are separated since not all nodes participate in staking. A node can run multiple validators, one for each 32 ETH they have staked. Each validator handles the cryptographic functions for its attestations and block proposals. The execution client used to be the only client required for PoW Ethereum, but with PoS it only handles transactions and mining has been turned off. The execution client updates the state and gossips transactions to other nodes in a peer-to-peer network separate from the beacon clients which have their own peer-to-peer network to gossip blocks and attestations [ge23]. During operation, the beacon client sends

transactions from new blocks to the execution client to check their validity, and if the node is scheduled to propose a block the execution client sends pending transactions to the beacon client to be sent to the validator client to use in the block. The beacon client without a validator can keep track of the canonical chain. The most common execution client is Geth by a wide margin, while the most popular beacon + validator client is Prysm followed closely by Lighthouse.

1.3.1 LMD GHOST & Casper FFG

An attestation is made of two votes, Casper FFG and LMD GHOST which are combined into a protocol known as Gasper [BHK⁺20]. The first block of each epoch is known as an Epoch Boundary Block (EBB) or checkpoint. If there is no block in the first slot of an epoch then the EBB is the latest block before that slot, and a block can be an EBB for multiple epochs if there are many missing blocks. A Casper FFG vote is a vote for the current and last EBBs, called the target and source respectively. When 2/3 of active validators by staked amount have voted for an EBB, that EBB becomes justified when the epoch it is in ends or if it has already ended. Furthermore, all blocks before the EBB are also justified. When the EBB after it also becomes justified, that first EBB and all blocks before it become finalized. The genesis block, or first block in the blockchain, is assumed to already be finalized. This is PoS Ethereum’s replacement for the n blocks deep confirmation rule and is called Casper the Friendly Finality Gadget (FFG).

An LMD GHOST vote is a vote for the current head, or latest block of the blockchain. LMD GHOST stands for Latest Message Driven Greedy Heaviest Observed Sub-Tree and is the fork choice rule replacing the longest chain rule from PoW. Let M be the set containing the single latest attestation from each active validator weighted by their stake. The canonical chain head is determined for a given node’s view of the entire blockchain, which is a tree. Each block in that tree is assigned a weight based on the tree and M . The weight for a block is defined as the sum of the attestations in M that are attesting to that particular block, if

any, in addition to the sum of the attestations in M that are attesting to any of the block's descendants. The head is then found by starting from the genesis block and traveling down the tree, picking the child with the most weight at each fork (ties are broken by the block's hash) until a leaf node is reached. The path taken is the canonical chain and the last leaf block is the head of the chain. A validator whose turn it is to attest submits this head as their LMD GHOST vote. In actuality Gasper uses a slightly modified version of this known as HLMD GHOST where the H stands for hybrid. In HLMD GHOST, for each node, instead of starting from the genesis the graph is truncated to not include any blocks before the last justified EBB block from the perspective of the start of the latest epoch (so new Casper FFG votes that may cause a new EBB to be justified don't affect the EBB being used by HLMD GHOST until the end of the epoch, it is frozen until the next epoch).

1.3.2 Slashing & Inactivity Leak

Similar to PoW consensus which required significant investment in hardware to carry out a successful attack, PoS also requires significant capital to perform an attack and this capital is also destroyed unlike PoW through slashing or penalties. Slashable offenses include proposing twice for the same slot, attesting to different head blocks for the same target and source, and other offenses that similarly show self-contradictory behavior. Most slashable offenses require a validator to have sent different messages to different parts of the network, so the protocol may not be able to find this behavior on its own. Therefore, in order for a validator to be slashed a block proposer must include a report in their block and for the effort required to identify slashable offenses they are compensated with a whistle blower reward, further incentivizing security. When a validator commits a slashable offense, it receives an immediate penalty of $1/32$ of its stake and is queued to exit the set of validators involuntarily. Halfway through this queue the validator receives another penalty corresponding to the number of other validators being slashed, which disincentivizes large attacks on Ethereum's security. Slashed validators receive no rewards and also receive missed attestation penalties while they are in the queue

[Edg23].

Finality requires $2/3$ of validators by stake to vote for an EBB. In a disaster scenario where more than $1/3$ of validators go offline blocks would stop being finalized. To address this, if the Beacon chain hasn't finalized an EBB for more than 4 epochs it enters an inactivity leak mode. During this mode, attestations stop being rewarded and missed attestation penalties grow quadratically with time until enough stake of inactive validators has been destroyed that the active ones have a $2/3$ majority by stake again.

1.3.3 RANDAO

It is necessary for block proposers to be chosen at random for fairness and security. If they were not random, an attacker could predict who would be a proposer far in advance and organize an attack against them or collude with them. Ethereum brings a decentralized pseudorandom number generator on chain with a DAO called RANDAO.

Originally a RANDAO was created as follows [RAN22]. Participants send a transaction to the DAO contract depositing Ether and submitting the hash of a secret number, a commitment. After a particular time period they submit another transaction to the contract revealing the value of the secret and the contract checks that this value hashes to their earlier submitted value, otherwise it is discarded. After the participants secrets have been collected, the contract combines them with a mixing function and uses that as the seed to a PRNG, generating a random number. The contract distributes the random number to those that send a transaction asking for it and pay a fee. Those fees are distributed to the initial participants as a reward and they also get their deposits returned. If a participant didn't reveal their secret number, their deposit is divided among the participants that did.

If the last participant could see the others' secrets before presenting theirs, they could craft their secret to get a particular seed they want after it is mixed with the others. RANDAO is robust to this type of attack since it ensures that participants have to pick their secret value before they can know what value others have submitted since they cannot reverse

the hash of others' secrets. A participant could still try to influence the seed by strategically choosing whether or not they reveal their random number but failing to reveal their secret number comes at a financial cost. With RANDAO, randomness accumulates with each additional secret added to the seed, and even with many colluding participants the value generated can still be random.

In practice [Edg23], RANDAO is implemented slightly differently on the Ethereum Beacon chain. Participants are block proposers who already have stake deposited into the Beacon chain. The secret value simply needs to be something that is unpredictable but verifiable. Instead of hashing a secret random value and later submitting the secret, proposers simply sign the current epoch number with their private key and include that in their block in a field called the *randao_reveal*. The signature replaces the secret value, is easily verifiable with the proposer's public key, and is random based on the seed used when the proposer generated their key pair. Each *randao_reveal* value is mixed with previous values by hashing it and xor-ing with an accumulating RANDAO seed. If there is an empty slot the RANDAO seed is not updated for that slot. Once per epoch the RANDAO seed is used to generate a random number that picks the proposers and committees for the next epoch. A proposer can try to manipulate the seed by not proposing in their slot and while that can slightly bias the RANDAO seed, it comes at the cost of block rewards. Ethereum's contributors believe this small bias isn't enough to have an effect on security for consensus but it could be a problem if the RANDAO value is used as a source of randomness for the application layer.

1.3.4 Attacks on Consensus

An attacker may attempt attacks that result in a reorg, double finality, or finality delay. A reorg, or reorganization, is when an attacker forces a change to the canonical chain, this is easier for blocks that have not been justified or confirmed. The point of a malicious reorg like this might be to censor some transactions, double spend with a seller who decides a transaction is confirmed before it is officially finalized by the chain, or if they are scheduled to

be a proposer soon, reorder transactions so they can make additional profit, a process known as Maximal Extractable Value (MEV). In order to reorg blocks that have been confirmed, an attacker would need $2/3$ of the total stake to vote for a different EBB, which necessarily means they must control at least $1/3$ of the total stake which would be destroyed. Similarly, an attacker with $1/3$ of total stake may try to cause double finality where two forks are simultaneously finalized. Finally, an adversary’s goal may be finality delay, which is an availability attack where blocks stop being finalized for a period of time.

Ethereum’s protocol has made many of these attacks prohibitively expensive or recoverable such as through slashing and inactivity leak. Reorgs and finality delay attacks on a small scale can be more feasible. An ex-ante reorg is one in which an attacker tries to fork the canonical chain so that future, yet unknown blocks are removed. In one such attack, a malicious validator who is scheduled to propose at slot t keeps their block hidden until the next slot $t + 1$ in which they release their block as well as an attestation for the block from another validator they control in the previous committee (t) [SSNM⁺21]. With good timing and multiple adversary controlled nodes gossiping from different points in the network, the block can be released at the same time as the block from the honest proposer at slot $t + 1$. Both blocks would share the same parent but the adversary’s block has one attestation already and will be favored by the $t + 1$ committee. The next honest proposer at $t + 2$ would then build on the adversary’s block, leaving the honest proposer’s block at $t + 1$ abandoned. Ethereum updated to include a proposer score boost that boosts the LMD GHOST weight of blocks that were received on time to counter this attack [boo21]. An attacker would need many more malicious validators’ attestations to counter the boost an honest proposer gets from proposing on time, so it is still possible but less feasible. Another possible attack is a balancing attack. In this type of attack, an attacker tries to delay or prevent finality by causing a fork to happen with network delay or double proposing, then maintaining that fork perhaps by strategically attesting to blocks in each fork with malicious validators under their control. By maintaining an even split of honest validators on each fork, they can

prevent either from gaining a $2/3$ vote needed for finality. This attack is not as practical as the previous one since it requires having lots of precise information about the network and timing.

2 Introduction

PoS Ethereum is a highly complex interdependent system. Such complexity comes with a large attack surface; attacks can target multiple layers simultaneously with different goals. An attack may manipulate lower layers in ways that are not significant to that layer but that has downstream effects which disrupt higher layers. With this variability it is difficult to make guarantees about security from the protocol, and an experimental approach may be more favorable. We now look at some existing works that take different approaches to studying Ethereum experimentally through simulation and emulation.

2.1 Related Work: Ethereum Simulators

[HLR21] introduced an Ethereum Beacon Chain simulator based off of PoS Ethereum’s protocol specification code and documentation. It consists of simulated validators on a Beacon node and an event driven simulator environment. Events represent messages sent over the network and follow a stochastic model derived from PoS Ethereum’s protocol. The Beacon node’s full state including fine grained consensus data at the start of each slot can be logged, and simulation parameters can be specified at the start. The simulation parameters include functionality like scheduling a malicious validator that proposes twice in its slot. The simulation supports up to a few hundred validators on consumer hardware before performance becomes an issue and runs may not be repeatable due to parallelization race conditions. The simulator’s input parameters were tuned with data collected from a real Lighthouse Beacon chain client over the course of 19 days, including processing delays (time between the start of a slot and when the new block and attestations begin broadcasting),

finality delay, and fork rate (number of reorganizations). The simulation was used to run a network of honest validators under normal and high latency and attacks with malicious validators. In the high latency scenario forking was observed. The malicious validator attacks included multiple different types of slashable offenses and a balance attack. For the balance attack a malicious validator proposes twice in a slot, which leads to them being slashed but both blocks are still kept in the blockchain, to split the chain into two forks. More malicious validators then strategically attempt to balance the attestation weight of each fork from the honest attestations with their votes (each malicious validator only votes for one side of the fork so the protocol doesn't view their votes as slashable) so that the blockchain stays split for as long as possible.

While this simulation platform has its merits, there are some important drawbacks to consider. Data collected to tune the simulation parameters might not include outliers. For instance, the author could not confirm they ever saw finality delay occur in the 19 day data collection period and so set that parameter to zero. Furthermore, the simulation platform was released late 2021 with the data collection occurring between July and August of 2021. PoS Ethereum underwent one major update between the data collection period and the paper's release and less than three years later has undergone two more major updates. The simulator needs to be modified and retuned to stay up to date with Ethereum. The paper demonstrated some attacks that were possible with the simulator but if a researcher wants to use the platform for testing any malicious behavior not already implemented, it would not be possible without editing the simulator. In a similar vein the simulator is missing any simulation of or connection to the execution chain (so it is also not able to simulate application layer smart contracts) as well as simulation of the underlying network protocol.

2.2 Related Work: Container Based Ethereum Emulation

[GTV⁺19] used a container based emulator to run a private network of multiple PoW Ethereum nodes. While they did not use the emulator to evaluate any attacks, they char-

acterized PoW Ethereum’s performance under different network topologies and sizes. In their evaluation they used metrics such as the number of orphaned blocks and blocks with multiple descendants which are measurements of wasted computation and forking. Their emulation used the Geth client but replaced mining with a statistical model, making the client partly a simulation. The emulator’s underlying Internet network also appears to be more of a simulation since it uses a fixed routing policy rather than emulating BGP.

A different paper proposes a different emulation platform, Containerchain [GLQ21]. Containerchain combines Mininet, a network emulator which uses OpenFlow capable Open vSwitch network software, with Docker containers for virtual hosts. Users write a python script which defines the network topology and hosts on the network. Each host has its own docker container which runs an Ethereum client (Geth). The platform is a full emulation of both the network and Ethereum nodes. Docker images used to create the containers must be edited manually or the instantiated containers must be given commands while they are running to install custom software onto the nodes. This can be burdensome for custom applications that want to use this emulator for testing, some premade images are provided but they only offer support for PoW Ethereum through Geth. The emulator was also not used to test any attacks.

3 Proposed Design

In this thesis we aim to build a platform primarily for studying consensus layer attacks on Ethereum. In particular we place emphasis on support for attacks that use the Internet infrastructure which Ethereum is built on top of as an attack vector. The motivation behind this focus is that Ethereum has been carefully crafted so that directly attacking its protocol forces attackers to use the protocol’s own currency to gain influence, which can be slashed. Direct attacks like this therefore typically come with a high economic cost and that investment is destroyed unlike the case with PoW where an attacker would still keep their

computing hardware for future attacks, making these attacks mostly infeasible. Ethereum’s protocol has less control of the Internet network it is running on top of, meaning an attack at this layer can be more effective.

3.1 Objectives

We aim to create a set of tools to facilitate setup and analysis of an Ethereum emulation and attacks on the network. For our platform, we want virtual nodes that run real world Ethereum clients and software emulations of network devices. We choose a platform of virtual nodes over physical network hardware so that it is more portable, configurable, and low cost. We build on top of the SEED Internet Emulator, a platform which itself builds on Docker containers to emulate network devices and hosts. The SEED emulator has built in functionality to install widely adopted PoS Ethereum clients, Geth and Lighthouse, on hosts in the emulated network and connect them together, which we use. We modify the SEED emulator and add software to the emulation to suit the needs of performing and recording the state of the network under attack. We evaluate our new infrastructure with two simple network attacks: a partition attack and a network delay scenario. Following the execution of these attacks collected data is visualized and we analyze Ethereum’s response, drawing conclusions about the security of its consensus. This platform can be used not only for researching attacks and mitigations, but also for hands on teaching and lab activities as was one of the goals of the SEED emulator.

3.2 Contributions

Our contributions primarily take the form of three pieces of software: a data collection daemon, setup program, and a controller. The daemon continuously collects consensus data within the emulated network and the setup program accepts arbitrary AS-level topologies, creating the emulation based on it using SEED. The controller is the main contribution and has functionality for controlling the emulation while it is running, performing network level

attacks, retrieving collected data, and visualization tools for collected data such as beacon and execution node peer plots and interactive plots of the tree of blocks. We modify the SEED emulator to configure Ethereum clients to record more information about the state of the blockchain and events, and use the latest versions of those clients.

Our threat model for the partition attack is a malicious AS which can delay or drop connections, and spies at multiple locations in the network, nodes that act honestly according to Ethereum’s protocol but relay real time information about consensus to the attacker. For the network delay attack we consider a scenario with multiple compromised or impaired ASes, such as a global disaster or an advanced persistent threat (APT).

3.3 Emulation Over Simulation

Simulation refers to a system that closely approximates a real world system while emulation refers to a system that is exactly equivalent to a real world system. Simulations vary in how accurately they approximate the real world system, where an ideal simulation would be indistinguishable from the real system in the aspects that it is being tested or examined on. Generally, simulations come at a reduced complexity or computation cost with the ideal simulation replacing difficult calculations with equivalent simplified ones. Furthermore, simulators typically allow for repeatable runs through a seed given to its primary random number generator. An ideal emulation on the other hand doesn’t make these simplifications or approximations and simply is a copy of the real world system with its full computation cost. For this computation cost, the benefits an emulation provides are that they are easier to implement since they don’t require gathering data about the real world system and coming up with approximations and one can be absolutely sure it exactly mimics the real world system in every aspect if it is set up the same with the same initial conditions.

For this work we choose emulation over simulation since it allows for accuracy, and adaptability. Ethereum is a complex system with many interacting components making it difficult to guarantee that a simulation is always accurate compared to its real world counterpart.

There are a multitude of examples of machine learning training in a simulation environment exploiting small inaccuracies that don't exist in the real counterpart, and sometimes this isn't apparent until it is tested on the real world system which results in wasted time and resources. With Ethereum constantly evolving it can become burdensome to adjust the simulation with each upgrade, thus a platform built on emulation automatically stays up to date with the mainnet. Furthermore, an emulation does not require knowledge about what will be tested on it, making it useful for not yet known further research interests.

Similar to simulation, there are levels of realism with emulation. The particular emulation we build in this work uses real world Ethereum clients but makes simplified assumptions about the Internet network, replacing specific routers with generic ones. This should be sufficient for attacks that make use of higher level network protocols, such as BGP hijacks, but is not a realistic enough emulation for attacks that make use of targeting particular network hardware devices which a researcher does not want to simplify. One option for these cases is the SEED emulator's "bring your own internet" mode that allows physical network hardware to connect to the emulation, but this is not the focus of this thesis.

3.4 SEED Internet Emulator

The SEED Internet Emulator is an open source python library used to configure and build docker containers to setup an Internet emulation [DZ22]. SEED provides classes for network elements such as ASes, routers, and hosts, which a user instantiates in a python program to specify the desired network topology, services running on it, and peering relationships (EBGP and IBGP). Based on the program, SEED automatically builds images for each container including software that should be installed on them and startup scripts that configure the installed software and connections between nodes. Each host or router in the network is given its own docker container running Ubuntu Linux, and routers use BIRD routing software to form virtual networks with other routers and hosts. SEED embraces modularity by using a layer based design with composition, render, and compilation phases. In the composition

phase, users specify the network topology and peering relationships at lower layers and applications running on hosts at higher layers. The layers are all independent from one another, referencing virtual nodes. This modularity allows them to be mixed and matched with other SEED emulations. In the rendering phase the layers are combined by binding the virtual nodes to assignments to docker images in an internal abstract representation, and in the compilation phase the images are built in docker from the representation either for a local machine or on Google Cloud Platform. Docker then handles launching and running the emulation. Docker images are also layer based with each layer contain files and applications. SEED uses a base Ubuntu layer and adds layers with additional software to be installed on the node. Containers can be instantiated from the images to run a virtual host that has its own file system and applications.

The SEED emulator supports three additional features: hybrid emulation, bring your own internet, and distributed emulation [DZW22]. With hybrid emulation the real internet can be connected to the emulation. A special AS is added to the emulation that announces a generic prefix and acts as a default route. If traffic is not routed to any node within the emulation it exits through this AS into the real internet. Similarly, with bring your own internet (BYOI), physical network interface cards (NICs) can bridge ASes in the emulation to real word network hardware. With distributed emulation part of the emulation can run on separate computers which are bridged together using a virtual private network (VPN) or physical network switch. The SEED emulator also has multiple canned components which can preconfigure hosts in the network to run applications like DNS, web hosting, and Ethereum clients, in addition to supporting installing and configuring custom software. With the Ethereum component, hosts are setup with Geth and Lighthouse clients. One host is a setup server which generates keys and the genesis block for the other nodes to kick start the blockchain (this is only a one time process at the start of the blockchain), and one or more hosts also run a boot node which is an HTTP server that allows Ethereum nodes to find each other on the execution and beacon layers (anyone can participate as a boot node

making it decentralized). Finally, the SEED emulator comes with a visualization tool called Map which displays network topology and traffic within an interactive locally hosted web app.

4 Implementation

We develop our Ethereum emulator in python using the SEED emulator library. The implementation consists of three programs: the setup program, data collection daemon, and controller.

4.1 Setup

The setup program accepts as input two text files: `ASTopology.txt` and `ASClients.txt`. `ASTopology.txt` is a comma separated configuration file containing the desired AS-level topology in a format similar to the CAIDA AS Relationships Dataset [LK14]. The exact format is shown in Table 1. AS1 and AS2 are strings that uniquely identify ASes. The delay value is in milliseconds and is a fixed delay for the connection between AS1 and AS2 while the peering relationship can be peer-to-peer (p2p) or customer-provider (cp) where AS1 is the customer and AS2 is the provider. `ASClients.txt` is shown in Table 2 and each row contains the number of Ethereum nodes that will run in a given AS.

AS1	AS2	Delay	Peering Relationship
...	...	...	...

Table 1: Contents of `ASTopology.txt`

AS	Number of Nodes
...	...

Table 2: Contents of `ASClients.txt`

The setup program first deletes old docker images and containers from the previous simulation it was used to build. It then forms a graph from the edge data in `ASTopology.txt`.

An AS that only connects to one other AS in the edge data is considered a stub AS, while one that connects to multiple is a transit AS. It is possible for two stub ASes to only be connected to each other and no other AS, known as an island since they are isolated from the rest of the network. The graph is used to construct the ASes with the SEED library according to those rules. Hosts are also added to the AS based on values in `ASClients.txt`. Each AS is given its own prefix and has one internal network for connecting its hosts and routers together. A stub AS will have only one router while a transit AS has one router for connecting to the stub ASes connected to it and one additional router for each transit AS it is connected to. These routers are connected based on the topology of `ASTopology.txt` and given a fixed delay if one was specified for the edge they represent. All connections between routers are facilitated by an Internet Exchange in the SEED emulator. Next, on the SEED EBGp layer object, the specified peering relationships between ASes are defined. After that, an Ethereum service layer is added and a number of virtual nodes equal to the sum of all the nodes in `ASClients.txt` are assigned Ethereum client software. One node is picked to be the Beacon setup node, another to be a boot node, and the rest validators with 32 ETH staked at genesis. The data collection daemon is also installed on each node. This point of the setup program (marked with a comment) is where smart contracts can be scheduled to be deployed if one wants to emulate the application layer (see the B06 example of the SEED emulator), but we don't use the application layer in this work. Finally, the virtual nodes are bound to virtual nodes from the lower layers and the SEED emulator rendering and compilation phases are run. The setup program automatically launches docker-compose to start the emulation.

Initially the Ethereum execution blockchain will run with Proof-of-Authority (PoA) consensus, which is similar to PoW except instead of having to solve a hash puzzle to decide on proposers, the network is permissioned, meaning only authorized nodes can propose new blocks at a fixed interval. The reason for this is that Ethereum spent years as a PoW cryptocurrency before switching to PoS, and new ETH was initially distributed during PoW via

block rewards. PoS obviously relies on there being ETH distributed which can be staked, so this initial PoA start allows us to manually distribute ETH and skip having to do any mining. The Merge is scheduled to happen after 12 blocks of PoA, after which the blockchain is using PoS for consensus.

4.2 Data Collection

The data collection daemon was installed on each Ethereum host in the emulator by the setup program. This daemon continuously polls it's local Lighthouse client's Beacon node API HTTP server, which provides consensus data through a RESTful API. The reason this daemon needs to collect data is that the Beacon node API only keeps data about the canonical chain accessible long term and log files are an incomplete picture. If an attack is emulated and the nodes reach consensus again after it is over, data about forks would be missing and it would not be possible to analyze many of the effects the attack had. The data collected include information about Blocks (hash, parent, slot, proposer, attestations in the block), the current weight of the tip block for use in the partition attack, scheduled proposers for the coming epoch, and committees. This data is saved in log files to be read by the controller. The default log files from Geth and Lighthouse were also made more verbose by adjusting the startup commands used by SEED.

4.3 Controller

The controller is primarily a collection of functions that can be used to control nodes and routers in the emulation, retrieve data from the containers and applications running on them, visualize collected data, and automatically perform some network level attacks. It comes with a small command line interface to specify parameters and run some of the important functions without having to write any code.

The controller supports sending arbitrary commands to each docker container and receiving the output. It can disconnect and reconnect routers as well as add delay to their

connections using the traffic control (tc) command which was a feature missing from SEED. It uses this functionality to carry out automated attacks such as the partition attack, where routers are disconnected to partition the Ethereum nodes when a new block is about to be proposed and reconnected when forks on each side of the partition have equal weight. The controller polls data from the daemon and Ethereum client logs to time the attack precisely. Further attacks can be easily designed using the controller’s functions as a toolbox. For instance, using the Geth command function, an attack can be programmed to execute when a smart contract event occurs.

Some visualization tools are also part of the controller. A particularly useful one is the peer plotting function, demonstrated in Figure 1, which plots the protocol level connections between peers for the execution and beacon layers. Another useful visualization tool is the interactive block viewer, demonstrated in Figure 2. This tool retrieves data from each node’s data collection daemon and compiles it into a plot of the entire blockchain. All of the nodes’ data is combined into an omniscient view of the blockchain and each node’s independent view is also plotted separately. The plot can also contain markers indicating where attacks began and ended, and are interactive web pages that users can pan and zoom through to intuitively view different parts of the blockchain history. When the plots are generated, each node’s beacon and validator log files from Lighthouse are also saved with the plots, so a user can explore logged events that might correlate with what they are seeing in the plot.

No transactions are being generated while the blockchain is running since we are solely focused on consensus but transactions can be created using the Geth command function and the wallets created by the SEED emulator.

4.4 Limitations

Being an emulation, a tradeoff is performance. Scaling the size of the network and number of nodes is challenging with the primary bottleneck being memory followed by compute. Individual docker containers run routers within an AS, and if too many hosts are connected to

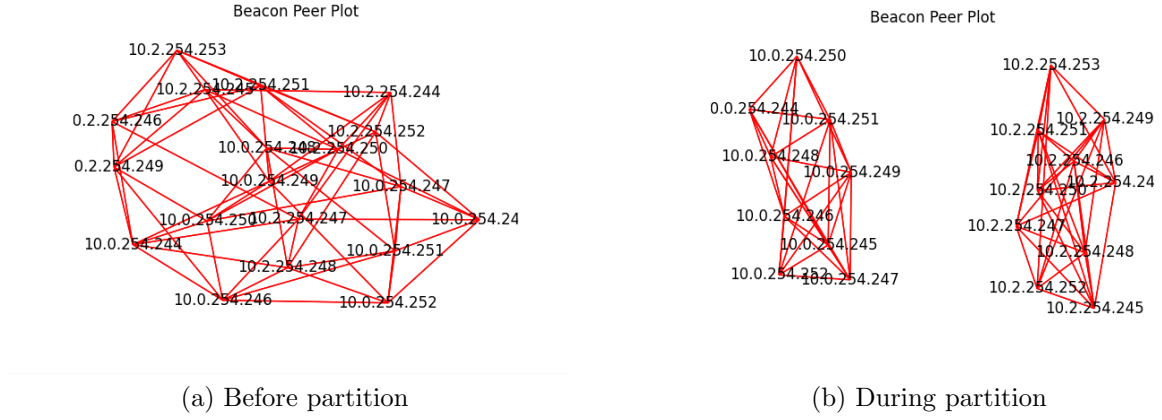


Figure 1: Consensus layer peer-to-peer connections plot during a partition attack

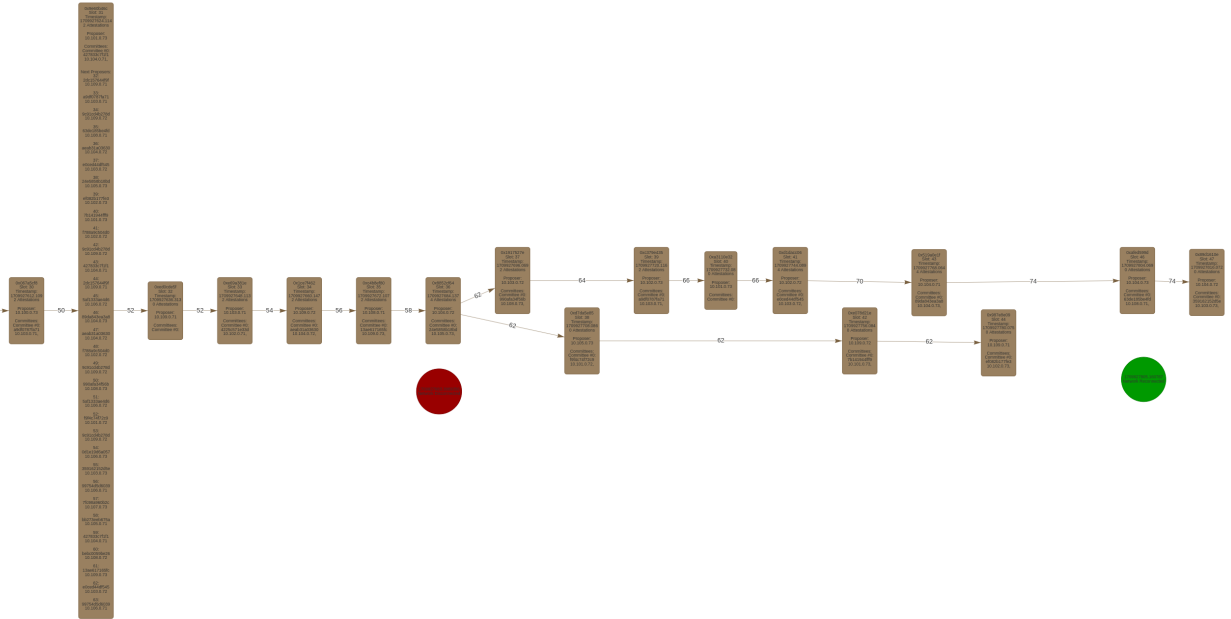


Figure 2: Interactive block viewer displaying the effects of a partition attack. Brown rectangles are blocks containing identifying information from each block's header and an arrow from that block's parent block. The tall block is an EBB which also lists future proposers, while the red and green circles correspond to when the attack started and stopped. The numbers on each edge indicate the weight determining which fork is more preferential to build on. The horizontal axis is time.

the AS, that router can become overloaded by the network traffic and associated processing. A solution to this is to make the workload more parallel by splitting hosts across multiple ASes which are connected together, with the rule of thumb being to have no more than 10 hosts in a single AS. Still, as the total number of docker containers increases, performance becomes an issue and larger simulations need to be given more resources. Performance issues don't just slow down the emulation, they also limit the size of the network because each node's Beacon node API HTTP server stops responding to requests which results in the daemon missing events. One method for decreasing the workload so that the number of nodes can be increased is to only have some of the nodes in a given AS running the data collection daemon as opposed to all nodes. This comes at the cost of reduced accuracy since one can't be sure every node within an AS always knows about every block other nodes in the same AS are aware of, for instance because they are not peers on the Ethereum beacon layer.

5 Results

We evaluate our platform as a tool for studying network attacks on PoS Ethereum's consensus by testing it on two attacks, a partition attack and a network delay attack. We test these attacks on a single workstation with 64 GB of RAM and an AMD Ryzen 9 3900X 12-core CPU. The emulator is run on Ubuntu 22.04 LTS in a VMWare virtual machine. We also test them with more Ethereum nodes on a high performance computer (HPC) hosted by CloudLab [DRM⁺19]. The HPC used is a single node with dual AMD EPYC 7543 32-core CPUs and 256 GB of RAM.

5.1 Partition Attack

Methodology The partition attack assumes a malicious AS which is the only connection between two sets of Ethereum nodes. The AS operator can choose when to drop and reestab-

lish that connection. The attacker has performed some network level observation or attack to put itself in this position, for example a BGP hijack, and while the SEED emulator has examples of emulating this, it is not the primary focus of our attack. The malicious AS operator is in control of some beacon nodes in each of the two sets of Ethereum nodes which relay it real time information about the current weight at the head of the blockchain from that node’s vantage point. The beacon nodes under the control of the attacker are only a source of information and do not act maliciously or participate in consensus.

The attacker’s goal is to disrupt Ethereum’s availability through delaying finality or ex-ante reorganizations. While they are in control of the choke point this is trivial, but an interesting question is whether they can use this leverage to cause a lasting effect on availability even after nodes establish an alternate route and the attacker losses their position of control. We model a scenario where the attacker creates a fork in the blockchain by dropping their connection and causing a partition in the network. They then time the reestablishment of the connection such that it occurs when each fork has an equal weight from the honest validators’ attestations, if possible. We observe whether this creates a lasting schism among honest validators with two metrics. The first metric is the number of blocks added, after the network is reconnected, to the fork that ultimately losses (becomes abandoned). The second metric is the number of missed blocks, or empty slots in both forks, after the network is reconnected.

We perform the attack by arranging the Ethereum nodes in a network topology where one transit AS is the only link between two sets of nodes, as shown in Figure 3. We instruct a router in this AS to drop its connection at the start of the attack and wait 75 seconds. Afterwards the router keeps waiting until both sides of the partition have equally weighted forks before reestablishing the connection. We model the attacker’s beacon node information sources by simply reading data from honest validators in the controller. This attack operates on the network level so an attacker could not be subject to any penalties by Ethereum’s protocol.

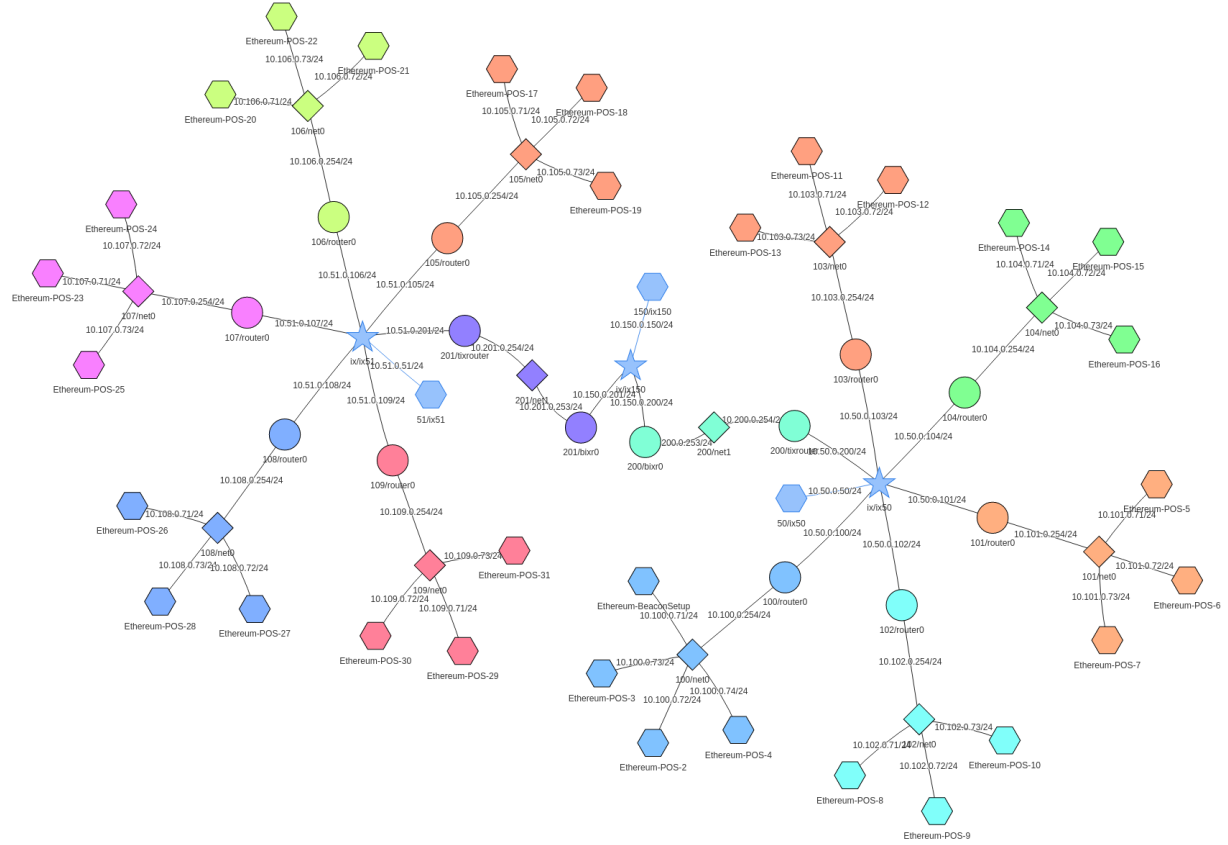


Figure 3: Partition attack network topology with 30 Ethereum nodes and a 50-50 split. 15 Ethereum nodes in stub ASes are connected to 15 other Ethereum nodes in stub ASes via two transit ASes. Nodes (routers and hosts) in the same AS are shaded the same color. The purple AS is malicious. Image generated using the map tool from the SEED emulator.

Observed Outcome Figure 4 and Figure 5 present the results of 10 trials of the attack measured by our two metrics on a 50-50 and 60-40 division of the network.

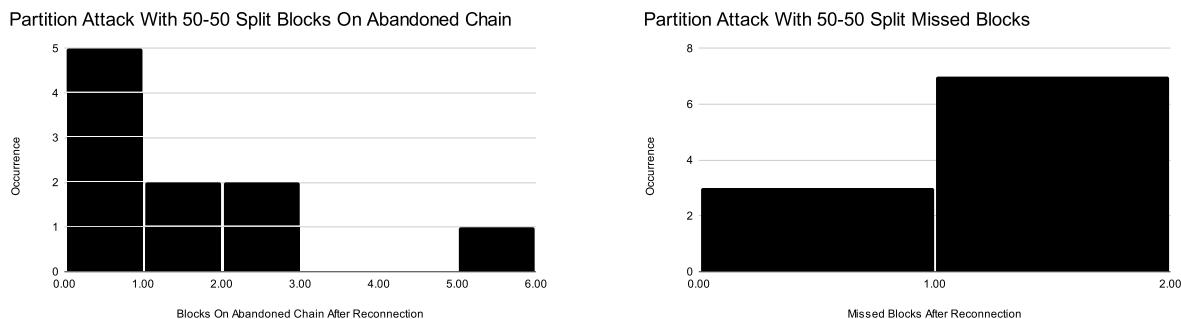


Figure 4: Histogram of blocks built on abandoned chain and missed blocks after reconnection. 10 trials with 30 Ethereum nodes and a 50-50 split of the network

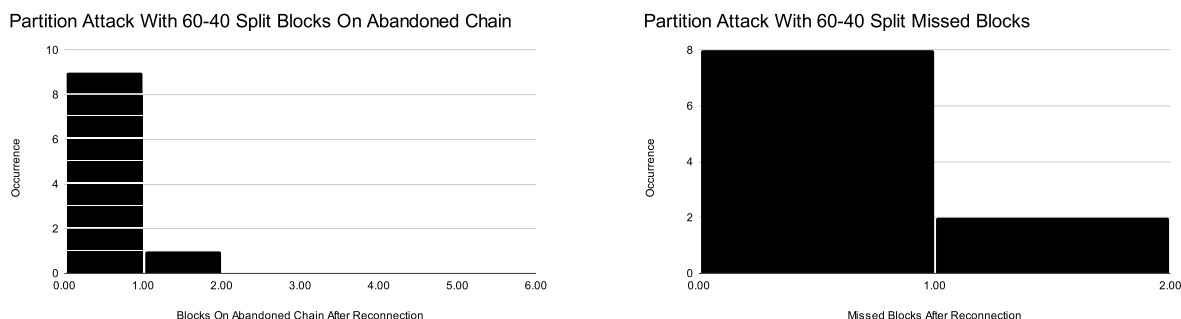


Figure 5: Histogram of blocks built on abandoned chain and missed blocks after reconnection. 10 trials with 30 Ethereum nodes and a 60-40 split of the network

Discussion Analyzing the data for the 50-50 case, we see that this attack sometimes creates a short term schism after the network is reconnected. On numerous occasions validators wasted their proposals on the wrong chain or missed their chance to propose if they switched forks in the middle of proposing or became desynchronized. The nodes were able to agree on one fork and abandon the other in all trials within one epoch.

Ethereum nodes decide on one of the forks through randomness leading to greater LMD GHOST weight on one fork: from RANDAO committee assignments being slightly uneven, the peers beacon clients are connected to and other sources. This means that with low

probability it may take quite a few blocks to come to a consensus on the canonical chain as we observed in the data. The next proposer after reconnection is not what decides on the fork. In our experimentation we observed the slots immediately following the time the network is reconnected to be on the abandoned chain in some trials and the winning chain in others. We also saw blocks added to the abandoned chain after multiple blocks had been added to the winning chain and only the winning chain since the point of reconnection.

We don't call this a balance attack because the attacker is not actively taking any action to maintain the partition and only relies on honest nodes to build on both sides of the fork. Considering the schisms we recorded in this passive approach, this attack may be greatly improved by turning it into a balance attack. One way to accomplish this would be allowing the attacker to control some validators that try to actively maintain equal weight on both chains, either with carefully timed legitimate attestations or slashable attestations at a financial cost.

For the 60-40 case, consensus on a fork is achieved much quicker. This is likely because the partition with 60 percent of validators initially views its fork as the canonical one and will continue to attest to it. Its greater attestation weight quickly sways the LMD GHOST fork choice rule of validators in the minority partition to adopt the 60 percent chain. Consequently, compared to the 50-50 split case, this unequal distribution makes running the experiment more difficult. About half of the trials needed to be discarded due to the two forks never being able to reach an equal weight, with the majority fork increasing in weight faster than the minority.

5.2 Network Delay Attack

Methodology The network delay attack evaluates Ethereum's resiliency to high latency. Ethereum's stated vision is to operate at a global scale, and it is not implausible to consider a scenario where a global disaster has occurred and multiple ASes are impaired. We model this scenario as an increase in the latency of all network connections. To emulate this global

disaster, we start with a topology that models the mainnet Ethereum network at a smaller scale based on the world topology from [GTV⁺19]. Each stub AS represents a country, with the number of Ethereum nodes in that stub proportional to the number of Ethereum clients of the mainnet Ethereum network present in that country as reported by ethernodes.org [etha]. These stubs connect to transit ASes, one for each country, that form the global Internet backbone and are fully connected to one another. We limit the network to the top 6 countries, and for the US and Germany we split them into multiple stub ASes connected to their transit AS due to the AS size limitation discussed in the Implementation. The latency between transit ASes is set to the average latency between those countries’ capital cities as reported by wondernetwork.com [Won] (the maximum latency in this data is the Finland and Canada connection at 110 ms with the minimum latency between France and the UK at 8 ms). The topology is shown in Figure 6.

For the attack we multiply all the latencies in the network by a scalar constant for a period of 60 seconds and observe the effect on fork rate and missed slots. The fork rate metric is defined as the total number of extra descendants on blocks with multiple descendants normalized by the number of countries in the topology. Missed slots are slots which are empty on all forks during the attack normalized by the number of countries. We run the emulation with 43 nodes on a single workstation and with 86 nodes on a single node of a HPC.

Observed Outcome We vary the scale factor and network size and plot against the fork rate and missed slots metrics defined in the methodology. The results are shown in Figure 7 and Figure 8.

Discussion The data show that Ethereum has resilience to large delays. Consensus can still be reached with multiple seconds of latency on connections and stakers are able to claim block rewards with minimal empty slots. Consensus begins to break down when latency rises to tens of seconds and continues to destabilize past that point. After the attack was

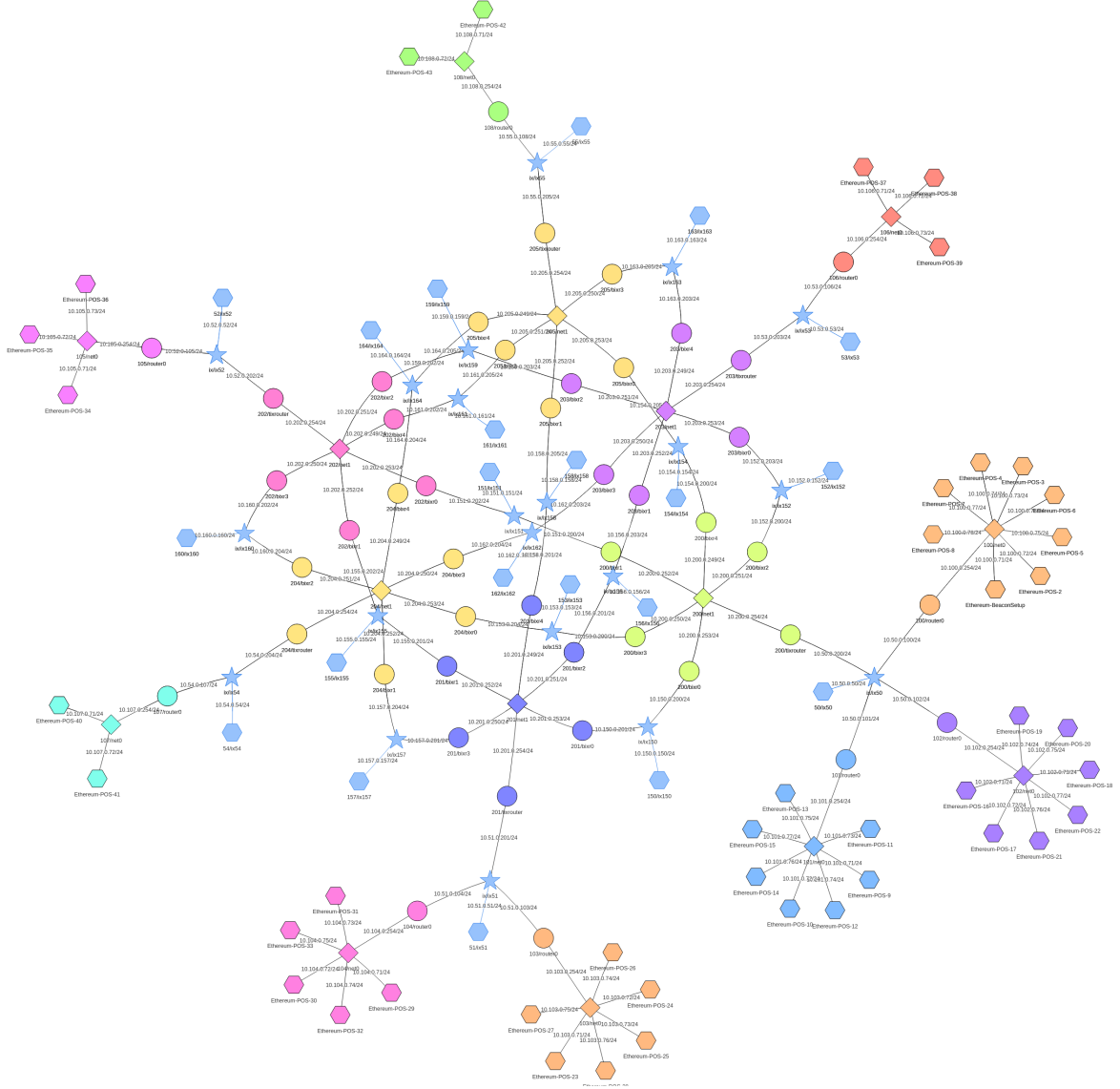


Figure 6: Delay attack network topology with 43 Ethereum nodes. The top 6 countries, by number of Ethereum nodes, are included in the network: the US with 22, Germany with 11, France and the UK with 3 each, and Finland and Canada with 2 each. Nodes for the US and Germany are split into multiple stub ASes for performance. Image generated using the map tool from the SEED emulator.

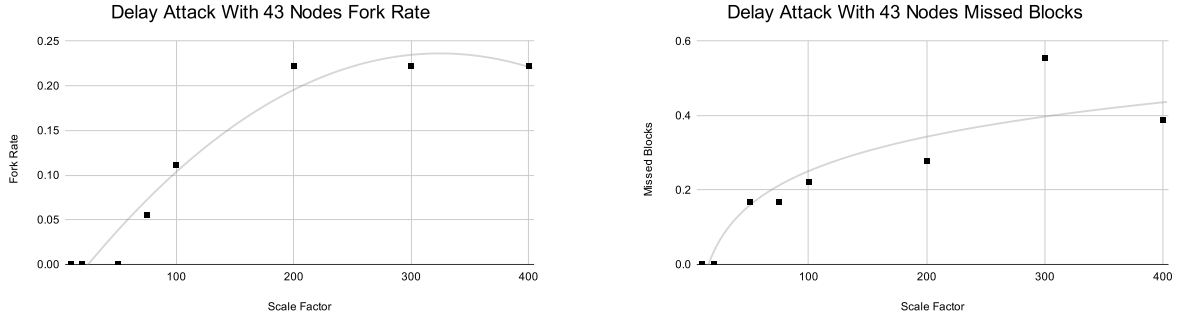


Figure 7: Delay attack results with 43 Ethereum Nodes. Each data point is the average of three trials.

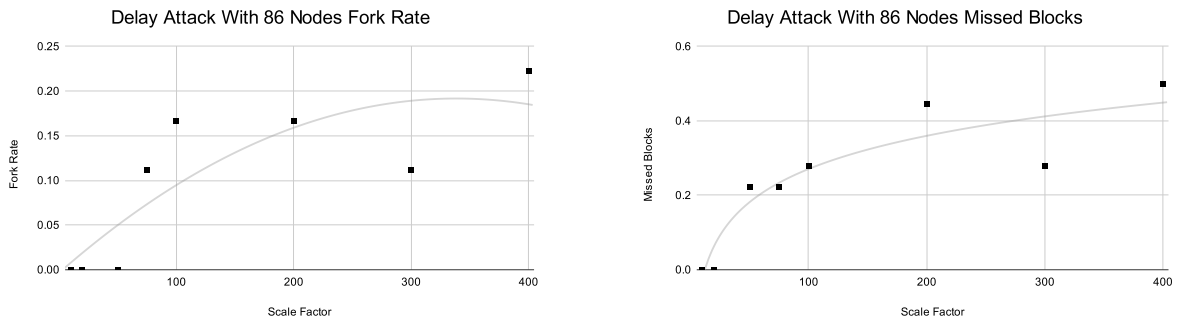


Figure 8: Delay attack results with 86 Ethereum Nodes. Each country's node count is doubled from Figure 6. Each data point is the average of three trials.

over and latency was returned to normal levels, consensus was quickly reestablished within 3 slots on all trials. High latency causes forks because proposers don't learn about the latest block to set as the parent of their new block in time and missed blocks occur due to reorgs and late changes to the canonical head block (also known as the tip block or the latest block in the canonical chain). The fork rate and missed slots data measured were about the same in the 43 node network and 86 node network cases which indicates that, at least for this specific type of attack, the emulation performance imposed limitation on network size does not impede drawing useful conclusions about Ethereum's consensus.

6 Future Work

Further research can involve using this platform to identify and test novel attacks on PoS Ethereum. One possible direction is to look at network attacks targeting the application layer. After the Merge, the RANDAO value was made available to the execution layer and can be used as a source of randomness in smart contracts [Edg23]. However as was discussed in the Background, the RANDAO value can be subtly manipulated by choosing whether or not to propose a new block when scheduled to do so. Smart contract applications that rely on this source of randomness, such as lotteries, are thus potentially vulnerable to an attacker that has multiple validators manipulating the RANDAO value. Ethereum developers currently have no plans to address this flaw. Such an attack would not result in the attacker being slashed and although they lose out on block rewards, they may stand to gain more from their attack on the application layer. A custom validator client doesn't need to be implemented to perform this attack on our platform, it can easily be modeled by putting the attacker's validators in an AS and disconnecting that AS for a period when the attacker is scheduled to propose and doesn't want to update the RANDAO seed.

Another area of interest is an evaluation of inactivity leak. It may be possible to manipulate this feature through malicious actions on the Internet network with the goal of

destroying part of the total stake. Other improvements that can be made include adding validator rewards and balance information to blocks in the interactive block viewer and adding malicious validators that can perform slashable offenses such as double proposing and contradicting attestations as was done in [HLR21] by modifying the Lighthouse clients.

7 Conclusion

In this thesis we detailed the features and consensus mechanism of Proof-of-Stake Ethereum as well as attacks on it. We motivated the need for a PoS Ethereum emulation platform through an analysis of previous approaches to evaluating Ethereum’s behavior and security. We stepped through out design considerations and implementation for the platform, and we used it to test two network level attacks. Finally, we listed further research directions for novel attacks on Ethereum and how this platform can play a critical role in finding and evaluating them.

References

- [AZV17] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 375–392, 2017.
- [BHK⁺20] Vitalik Buterin, Diego Hernandez, Thor Kampefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining ghost and casper, 2020.
- [boo21] Proposer lmd score boosting. <https://github.com/ethereum/consensus-specs/pull/2730>, Nov 2021.
- [But14] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014.

- [DRM⁺19] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.
- [DZ22] Wenliang Du and Honghao Zeng. The seed internet emulator and its applications in cybersecurity education, 2022.
- [DZW22] Wenliang Du, Honghao Zeng, and Kyungrok Won. Seed emulator: an internet emulator for research and education. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks, HotNets ’22*, page 101–107, New York, NY, USA, 2022. Association for Computing Machinery.
- [Edg23] Ben Edgington. Upgrading ethereum. <https://eth2book.info/>, Sep 2023.
- [etha] Ethereum mainnet statistics. <https://ethernodes.org/countries>.
- [ethb] ethereum.org. Ethereum development documentation. <https://ethereum.org/en/developers/docs>.
- [ge23] go ethereum. Node architecture. <https://geth.ethereum.org/docs/fundamentals/node-architecture>, Aug 2023.
- [GLQ21] Saul Gill, Brian Lee, and Yuansong Qiao. Containerchain: A blockchain system emulator based on mininet and containers. In *2021 IEEE International Conference on Blockchain (Blockchain)*, pages 1–7, 2021.
- [GTV⁺19] Maximiliano Geier, Claudio J. Tessone, Marco Vanotti, Silvio Vileiriño, David González Márquez, and Esteban Mocsos. 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, 2019.
- [HLR21] Marco Hünseler and Kerstin Lemke-Rust. Simulating an ethereum 2.0 beacon chain network. In *2021 Eighth International Conference on Software Defined Systems (SDS)*, pages 01–08, 2021.
- [LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- [Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. May 2009.
- [RAN22] randao. <https://github.com/randao/randao>, Jun 2022.
- [SSNM⁺21] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum, 2021.
- [Won] WonderNetwork. Global ping statistics. <https://wondernetwork.com/pings>.

Code

The code for this thesis is open source and available at:

<https://github.com/ronakstudios/seed-emulator>