Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Network-level Attacks Against Ethereum PoS

Bachelor's Thesis

Kai De Windt

dewindtk@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Lioba Heimbach, Tran Duc Muoi
Prof. Dr. Roger Wattenhofer

December 5, 2023

Acknowledgements

I am deeply grateful to my supervisors Ms. Heimbach and Mr. Tran Duc for their support and guidance throughout this thesis, and above all their patience during times of turmoil. I thank Professor Wattenhofer for providing me with the opportunity to undertake this research. Special thanks to Shana from the Flashbots team for their generous insights and directions in debugging the builder software and Nishant from the Prysmaticlabs team for their initial setup advice on the prysm beacon chain software. Their collective contributions have been crucial to the success of this work.

Abstract

This study delves into the feasibility of network-level attacks against Ethereum's Proof-of-Stake (PoS) consensus mechanism, specifically focusing on the Proposer-Builder-Separation (PBS) scheme. We simulate this environment on a custom development network and run two attack models. We find that in the new network landscape that PBS introduces, if a network attacker can successfully execute an attack that includes dropping or delaying packets sent between the relays and a validator, they are able to force the validator to build its own block, censor transactions, influence the profitability of certain blocks, and thus influence the outcome of the blockchain. We find that the attacker succeeds at a higher frequency with increasing drop rates and that adding a 500ms delay of outgoing packets from the relay suffices to get a 100% success rate.

Contents

Chapter 1

Motivation

After Ethereum's merge on the 15th of September 2022, although still in a state of research [1], the usage of the PBS scheme by validators participating in the Ethereum consensus protocol had risen to 80% in merely 3 months and has since kept this level consistently [2]. Prior collected data shows that there are clear centralization issues associated with the PBS scheme [2], raising concerns over potential vulnerabilities arising from them. Additionally, being one of the leading blockchain ecosystems today with a market cap of 265B$ at the time of writing [3], it is crucial to mitigate any potential risks associated with the foundational structures of ecosystems of such size. This research thus aims to contribute to the ongoing research of the Ethereum PoS PBS scheme in hopes of strengthening its network layer security.

Chapter 2

Background

2.1 Ethereum fundamentals

We first introduce the basic concepts surrounding Ethereum, PBS, and their network landscape, focusing on technical details revalent to the understanding of the different interactions between network participants, and ultimately the later discussed attack scenarios.

2.1.1 Blockchains and peer-to-peer networks

Blockchains

The Ethereum blockchain is a distributed ledger containing a chain of blocks. Each block holds transaction data and a cryptographic hash of the previous block.

Peer-to-Peer networks

In a peer-to-peer network, each participant is called a node, and communication between nodes does not go through central servers but occurs directly between their peers. This landscape ensures that the system remains operational even if individual nodes experience issues or disruptions. On a peer-to-peer network, the sharing of information occurs through a method called broadcasting, where a node holding information it wishes to share first shares this with its direct peers, who share this information in the same way reaching nodes all across the network. On Ethereum, the Consensus and Execution layers are run by two separate peer-to-peer networks, where consensus information and transaction data respectively are shared across the network in this manner (more in 2.1.2).

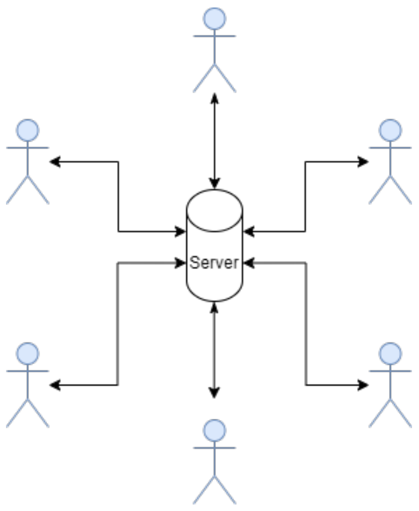


Figure 2.1: Server-Client network landscape

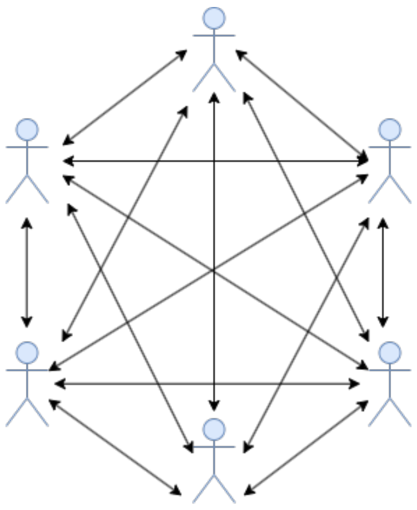


Figure 2.2: Peer-to-Peer network landscape

2.1.2 Proof of Stake: Execution and Consensus

On the 15th of September 2022, Ethereum completed its "merge," transitioning from PoW to PoS. The merge involved substantial changes to the network's underlying consensus mechanism, moving from a computationally intensive, energy-intensive PoW system to a more efficient and sustainable PoS system.

Execution Layer (EL)

The previously used PoW, which Ethereum had relied upon since its inception, was maintained by a peer-to-peer network of miners. Miners were responsible for accepting, verifying, and finally broadcasting transactions into the network while maintaining the states of the Ethereum Virtual Machine (EVM). Additionally, they competed against each other by solving computationally intensive mathematical problems to earn the right to propose the latest block in the chain. The winner of this competition would then group the transactions it has recorded through the broadcastings into a block, cryptographically signing the content with its private key, the results of which were broadcasted into the network yet again, earning itself a reward. Each miner would then record and store this latest block and the loop would continue.

This fundamental architecture still exists on post-merge Ethereum as the execution layer, consensus does however not take place on this layer anymore.

Consensus Layer (CL)

In PoS, the role of block proposals has been assigned to entities called validators. **Validators** are abstracted by wallets and are financially involved in the consensus protocol: to participate as a validator, one must deposit 32ETH (roughly 67'000$ at the time of writing) to a dedicated contract address, which would be locked up as collateral. This is a process known as "staking". They are then tasked with 2 primary roles (the complete implementation can be found in [4]):

- When selected, propose the next block.
- When selected, verify and attest to block proposals.

If a validator fails to complete the task they have been assigned, they lose a portion of their staked ETH. This incentivizes them to act truthfully and according to protocol. As to how these proposals and attestations are managed, this is where the Consensus Layer peer-to-peer network comes into play.
The consensus layer is yet another peer-to-peer network sitting above the execution layer. Nodes of this network are named "beacon nodes" and are typically (but not necessarily) managed by validators. Beacon nodes maintain the "beacon chain" where information about previous attestations and proposals are stored, as well as future proposal duties. When a validator commits to a block proposal or an attestation, it broadcasts this information to the consensus layer network through a beacon node. The selection of set validator duties happens through an algorithm called RANDAO, which selects these duties in direct relation to the amount of ETH validators have currently staked.

Definition 2.1 (Slot, Epoch)**.** A **slot** represents a fixed time interval of 12 seconds. For each **epoch** of 32 slots, validators are randomly selected by the RANDAO algorithm in advance to propose a new block to the chain. This block contains the latest batch of verified transactions that have been submitted to the execution network.

In summary, the consensus layer and execution layer are separate networks in Ethereum, but they are closely linked. These two node software are run in pairs by validators so that they can group transactions from the execution layer into a block, which they sign with their private key and then broadcast through the consensus layer.

2.1.3 Maximal extractable value (MEV)

MEV is an important concept when it comes to transaction grouping. Transactions sent into the execution network are not included on a FIFO (first in first

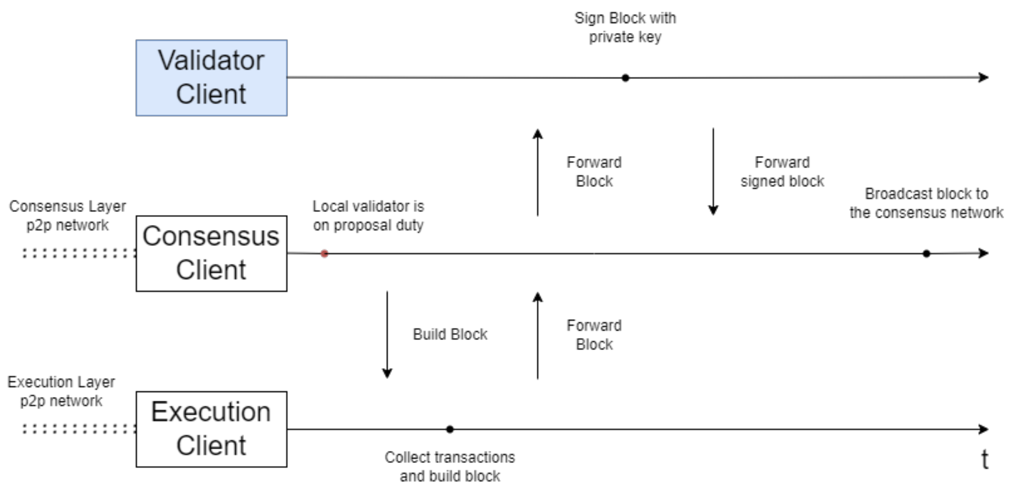

Sign Block with private key
Validator Client
Forward Block
Forward signed block
Consensus Layer p2p network
Consensus Client
Local validator is on proposal duty
Broadcast block to the consensus network
Build Block
Forward Block
Execution Layer p2p network
Execution Client
Collect transactions and build block
t

Figure 2.3: Timeline of a block proposal from a validator on PoS Ethereum. When the validator's proposal time arrives, the beacon node commands the execution node to group its transactions and build a block. This is passed to the validator client which signs this with its private key, committing its proposal. This is then broadcasted to the beacon chain, ready to be verified by its peers and attested to.

out) basis, but rather block builders have the freedom to choose exactly which transactions they wish to include in the block that they are building. This is often decided by the profitability of the transaction (how much priority fee the builder expects to receive from the transaction), but also from private transactions it wishes to include in the block at that time. Selecting the group of transactions from the mempool (local storage of pending transactions recorded through the peer-to-peer network) to form the most profitable block is known to be an NP-hard problem [5], thus is a computationally and algorithmically intensive process, a capability which not all miners possess. The profitability of a block is denoted as the block's **value**. Today, as we will see in the next section 2.1.4, a market of block builders has emerged dedicating themselves to optimizing MEV and offering their built blocks to proposers on duty.

2.1.4 Proposer-Builder separation

PBS is a design approach to decouple the tasks of block proposing and block building and the focus of this work. Where traditionally these tasks are performed by the same entity, such as a PoW miner or a PoS 3-client validator (see fig: 2.3), in this scheme, the validator is given the choice to either produce their block through their own builder execution client or take a block offered to them through a market of dedicated block builders. Through the natural dynamics of

MEV (section 2.1.3) and block profitability, validators are financially incentivized to build a block of high value and thus opt into the PBS scheme. Data collected since the merge and the launch of PBS shows that a validator on average earns more in block rewards using PBS as opposed to building their own block[2]. In this section, we lay out the key interactions between entities participating in this scheme.

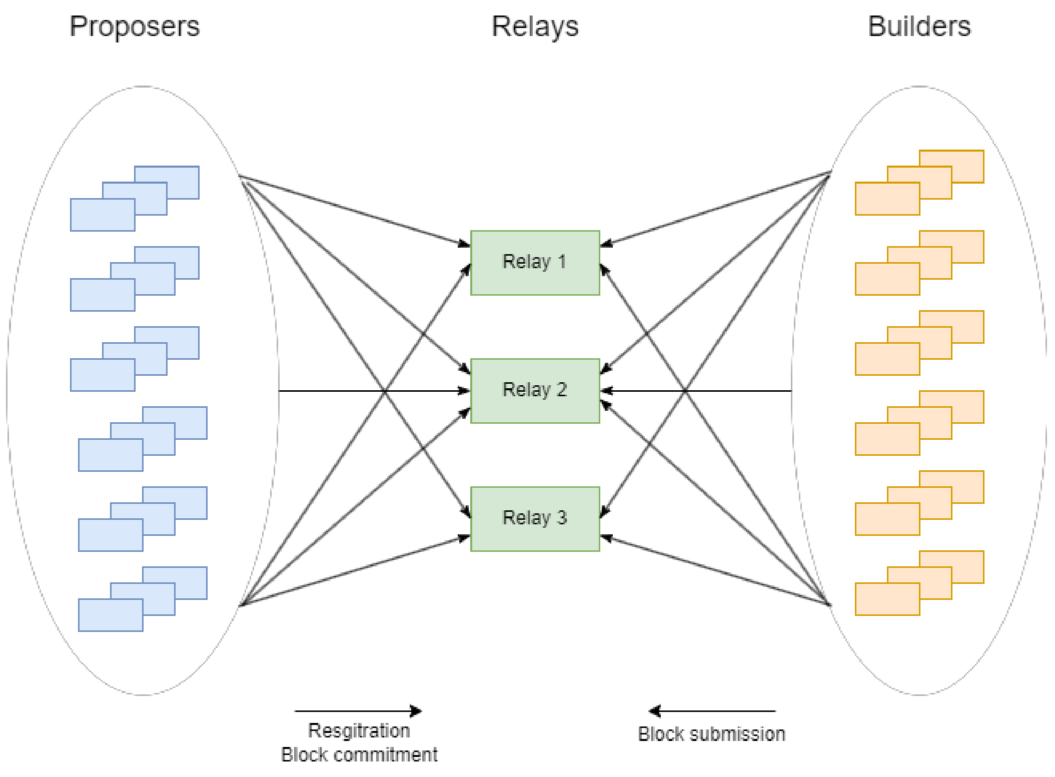


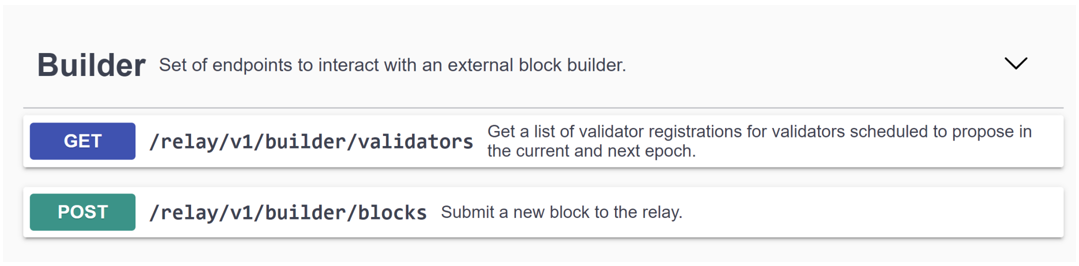
Figure 2.4: Network landscape of the PBS scheme. Block builders submit their blocks to relays, which are accessed by and forwarded to validators on duty.

An important component of the PBS scheme is the relay intermediaries. Relays are double-trusted entities that have the role of relaying blocks from the builder market to the validators on duty. They do so by exposing endpoints to both sides which can be accessed publicly.

Relay - Builder interactions

When a builder wishes to submit their block to a relay, it does so alongside a "bid", which is a transactional value in direct correlation to their produced block value, which will be transferred to the validator which decides to choose this block. This is done by builders appending a transfer transaction at the end

of the block. Relays take this information and are trusted to forward, to the proposer on duty, the block with the highest bid in its array of blocks it has received submitted.
Figure 2.5 shows more details about the standardized builder endpoints


Builder Set of endpoints to interact with an external block builder.

GET `/relay/v1/builder/validators` Get a list of validator registrations for validators scheduled to propose in the current and next epoch.

POST `/relay/v1/builder/blocks` Submit a new block to the relay.

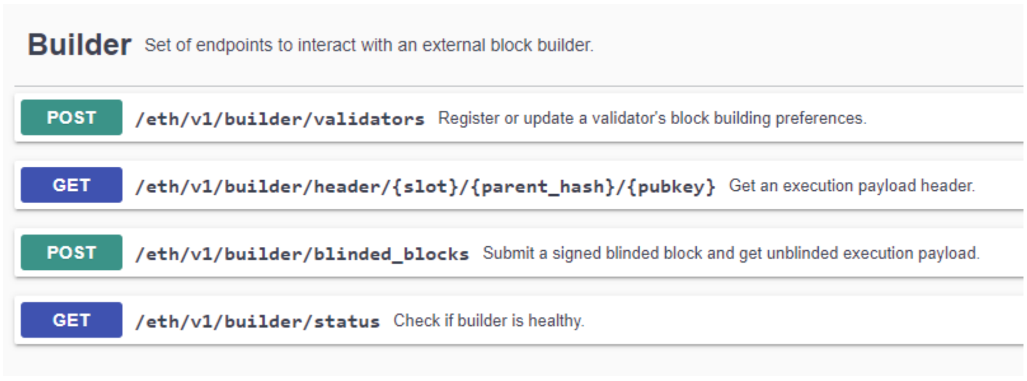
Figure 2.5: Relay: Builder API endpoints

Relay - Proposer interactions

When a validator wishes to opt into the PBS protocol, it first has to register itself to the relays it wishes to get blocks through. Once registered, the proposer may access the relay when it is selected for block proposal. This is done in a 3-step process:

- The relay verifies that it is the validator requesting the block is currently on proposal duty and exposes the highest bid value of the blocks it has been submitted.
- If the validator is content with the bid, It commits to the block by blindly signing the header without seeing the contents of the block. This is done to protect the value of the builders and no transaction reordering can happen from the validator's side.
- Once the commitment reaches the relay, the relay verifies this signature and broadcasts the block into the consensus layer, finalizing the slot. Only after this step does the proposer get to know the contents of the block. This process is typically managed by software such as mev-boost from the Flashbots research organization, open-sourced for the proposer community.

A visual representation of this process can be seen in figure 2.7. During this process, the validator trusts that the block it receives is valid, that the bid value it was promised is transferred to their address, and that the broadcasting occurs without issue. This makes relays' honesty and availability a critical factor to the function of the PBS scheme. The validator is however not forced to sign the block it receives.



Builder Set of endpoints to interact with an external block builder.

POST `/eth/v1/builder/validators` Register or update a validator's block building preferences.

GET `/eth/v1/builder/header/{slot}/{parent_hash}/{pubkey}` Get an execution payload header.

POST `/eth/v1/builder/blinded_blocks` Submit a signed blinded block and get unblinded execution payload.

GET `/eth/v1/builder/status` Check if builder is healthy.

Figure 2.6: Relay: Proposer API endpoints

2.2 Network Layer Attacks

2.2.1 Attack models

Numerous attack models pose threats to networks which we lay out in this chapter. Oftentimes, it is a combination of many of these primitives that form an attack to fulfill a malicious purpose to a network. As we explore potential vulnerabilities of the PBS scheme we acknowledge the danger each of these primitives hold and the intricate ways they may interplay to compromise the security and reliability of the scheme.

DDoS Attacks

Distributed Denial of Service (DDoS) attacks [6] can flood a network with a massive volume of requests, overwhelming the network's capacity to handle legitimate requests. In the context of PBS, relays that have public API endpoints exposed would for example be a target of DDoS attacks. Relay service providers are aware of this attack vector and enforce relays with measures such as priority queues [7].

Eclipse Attacks

In an eclipse attack[8], a malicious actor isolates a victim node from the rest of the network by surrounding it with adversarial nodes, effectively gaining control of the information it receives and sends.

Sybil Attacks

Sybil attacks[9] involve a malicious actor creating multiple fake identities (Sybil nodes) to gain a disproportionately large influence or control over the network.

Routing Attacks

Routing attacks[10] involve manipulating the routing tables to divert network traffic through malicious ASs and routers. One can for example introduce substantial delays to packets sent across networks by manipulating routing paths in this manner.

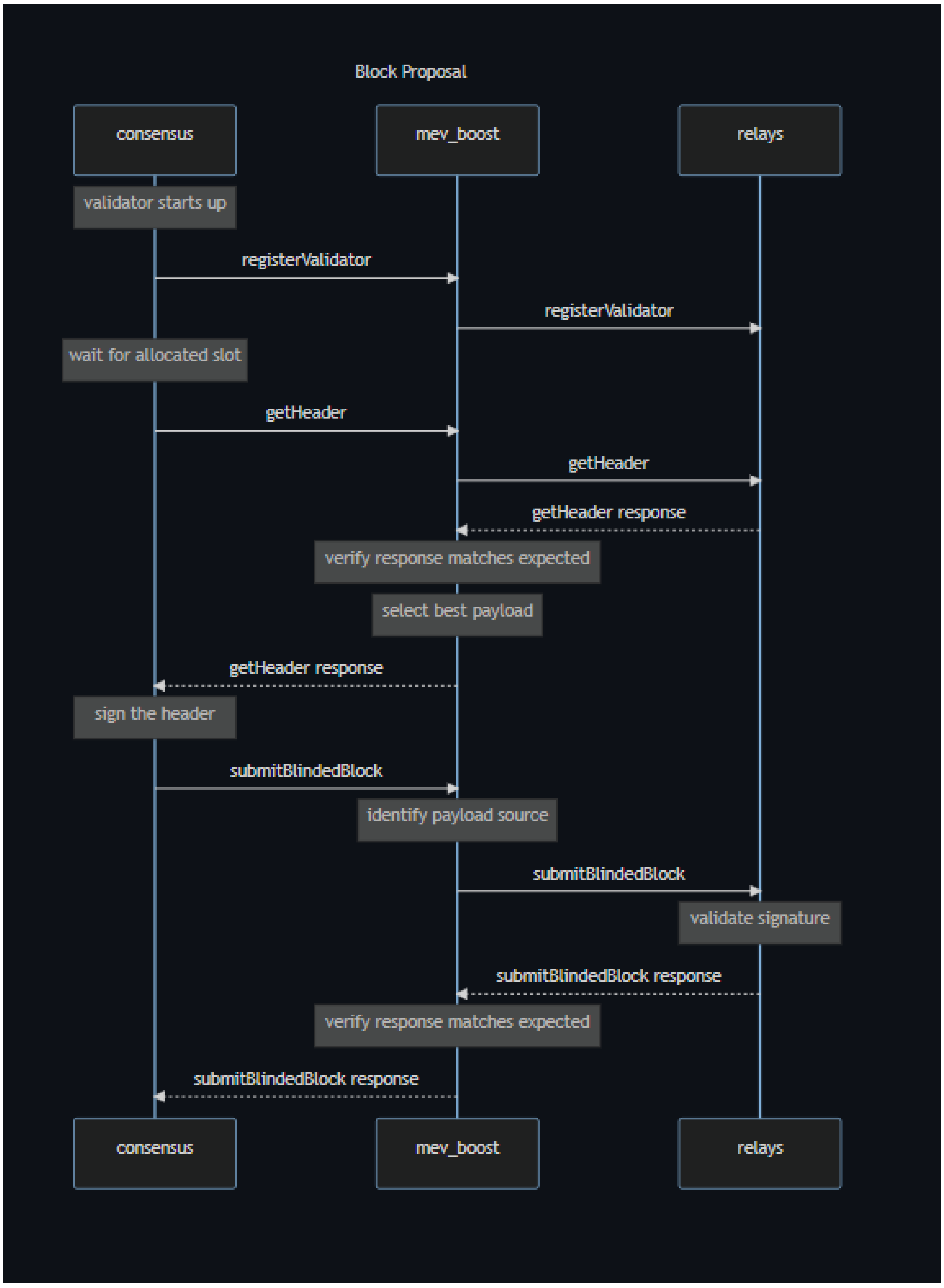


Figure 2.7: Block proposal flow through mev-boost: registerValidator, getHeader, submitBlindedBlock

Chapter 3

Simulation

In this chapter, we theorize potential ways one can attack the network landscape that the PBS scheme has introduced. We back these by simulating the attacks on a small custom development network using standardized and open-sourced software, widely in use on Ethereum mainnet at the time of writing [11]. These include Flashbots research organization's mev-boost [12], mev-booost-relay[13] and boost-builder [14], Prysmaticlabs' Prysm consensus and validator clients [15] and Ethereum's Go-Ethereum execution client [16]. The simulations were conducted using default network configurations (timeout values) and submission offsets.

3.1 Setup and Environment

Docker

For the simulation, custom Ethereum development networks were set up using Docker [17]. Docker is a development tool that allows the encapsulation of applications and their dependencies in so-called containers, which can be placed in isolated IP networks and given addresses of choice. This way, an environment close to reality can be created, where standalone apps and services run independently communicating with each other over their intended protocols.

Traffic control

The network attacks have been realized by using Linux traffic control (tc) [18]. TC is a command-line tool that allows users to configure and manipulate the traffic control settings for network interfaces. It is used for controlling the traffic flow, shaping, prioritizing, and controlling bandwidth on a Linux system. Operationally, when using tc, one defines traffic classes, assigns filters to classify packets into these classes, and then applies disciplines to control the transmission behavior of the queues associated with each class.

In our development networks, we simulate the following entities: a proposer, relay services, and builders. The proposer will be referred to as the "victim". Each entity has access to a pair of Consensus and Execution nodes, which fulfill their roles as CL and EL peer-to-peer network participants, forming the Ethereum network (more in section 2.1.2). Additionally, the victim has access to a mev-boost service. Each entity and its interactions are described below.

Proposer/Validator

The proposer consists of a validator client, a consensus client, an execution client, and a mev-boost instance. The validator client is responsible for managing the proposer's private keys for block commitment signings. In our case, it manages 64 different proposer keys which represent the genesis proposers wallets. This number was chosen to have the effect of diversification to proposer addresses selected by the randao selection (more in section 2.1.2) as opposed to having a single proposer array ignoring one of the purposes of the beacon network. The validator client is connected to its consensus client via RPC, which is itself connected to its execution client and the mev-boost instance. Each block that is proposed and broadcasted through the consensus node is first passed through the validator client to be signed by the current proposer wallet private key. This is the only validator client instance present in the simulated devnet. The rest of the nodes' functions and roles are laid out in sections 2.1.2 and 2.1.4. The mev-boost instance is responsible for interactions with the relays and manages queries and responses.

Relay

The relay server consists of several database containers, a housekeeper service, an API service, as well as its consensus and execution clients. The housekeeper service is responsible for periodically updating the database with the active proposers, future proposer duties, and validator registrations, by accessing its local consensus client. Additionally, it updates the statuses of its connected builders. The API service exposes the relay API endpoints and handles its calls (more in section 2.1.4). When a builder submits a block to the API service, the block gets passed to the execution client for simulation and validation first, then verifies that all fees are transferred correctly to the validator address before passing the header to the mev-boost service.

Builder

Builders are a set of consensus and execution clients that build and send blocks to relays on every occasion. On a new slot, the consensus client calls the execution client for block building, which directly sends this data to the relay.

The goal of the following simulations is to attack the specific connections between relays and the mev-boost instance, potentially restricting the victim's access to builder-built blocks and influencing the outcome of the blockchain.

3.2 Scenario 1: One Relay and one Builder

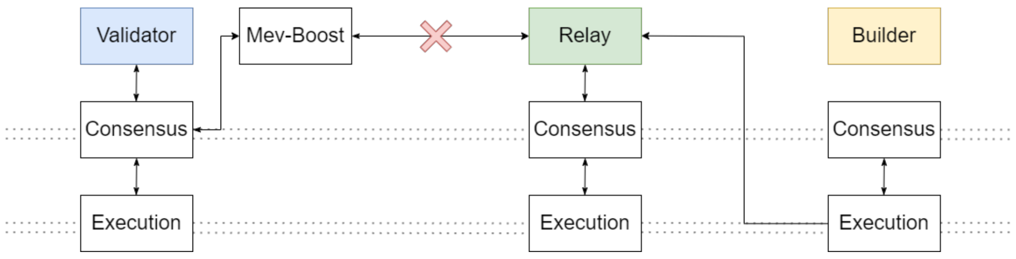


Figure 3.1: Simulated Devnet consisting of 1 proposer, 1 relay service, and 1 builder. The connection with the red cross will be restricted.

Figure 3.1 depicts the simulated network. We define the following attacks and theorize their outcomes:

- **Scenario 1, Attack 1**: Drop a percentage of OUTGOING packets from the relay to the mev-boost service. We theorize that the victim proposer falls back to its own block building, in ratio to the amount of packets dropped.
- **Scenario 1, Attack 2**: Delay OUTGOING packets from the relay to the mev-boost instance. We theorize that there will exist a cutoff point where the delay causes the victim proposer to fall back to its own block building.

We investigate the effect of these attacks starting with the network in a default condition and linearly increasing the degree of traffic control.

Siulation procedures

First, we simplify the network conditions in the following ways:

- **We keep the general transaction pool empty**: this is the default condition we simulate the network in. Each consecutive block is empty until we populate the transaction pool.

- **We populate transactions only through the builder execution client**: If we record a transaction included in a block, we know that this came from the builder, through the relay server and the mev-boost service.

Then, we:

- **Continuously send transactions into the network through the builder**: we use a script to listen to new server-sent events (SSE) sent by the builder's consensus client interface (specifically, the 'block' event. More details in [19]) and then on each new slot, populate a simple transaction sending an amount of Ether from a wallet we own to a random address. This wallet needs to be given Ether on the network setup in the genesis configuration files of the blockchain.
- **Apply traffic control to the mev-boost-relay API service container**: we define a filter to the root queueing discipline of the container's network interface, filtering IP packets with the destination IP address of the mev-boost container.

We simulate 100 blocks for each level of traffic control and record the transactions included in these 100 blocks through the victim's consensus client interface.

Definition 3.1 (Attack success rate)**.** We define the attack's success rate as the ratio between successes and failures, where an attack is considered successful if and only if the block is empty of transactions.

3.3 Scenario 2: Network of two Relays and two Builders

Figure 3.2 depicts the simulated network.

Simulation procedures

In addition to the network conditions set in Network 1 (see conditions 3.2), we change how transactions are populated:

- **We populate transactions of higher value to the builder1**: By populating transactions with higher priority fees (fees directly sent to the builder address) at the builder1, the **bid** of this builder's submitted block increases, causing the mev-boost to chose these blocks over the ones submitted by builder2.

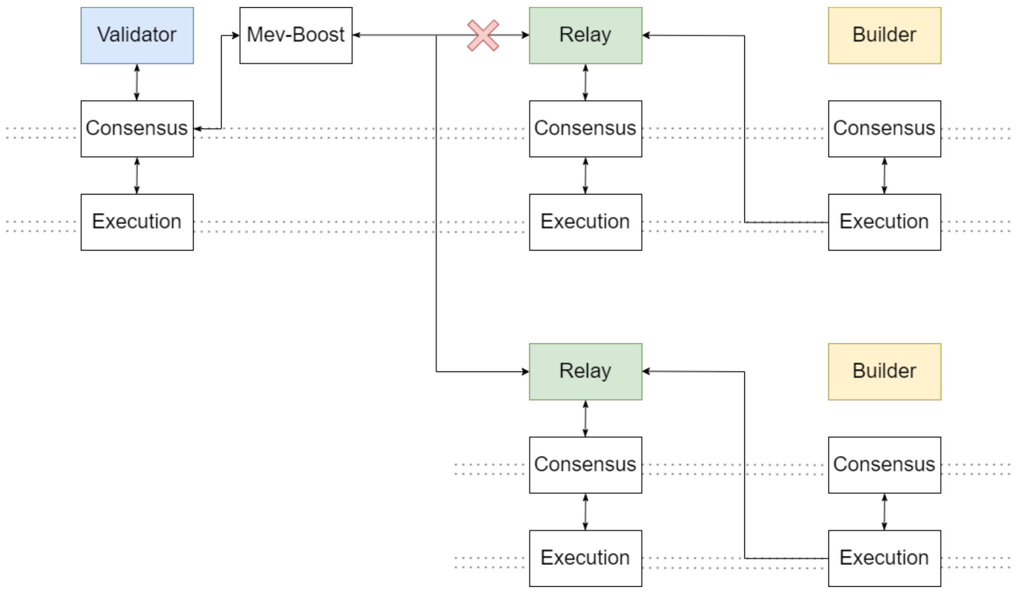


Figure 3.2: Simulated Devnet consisting of 1 proposer, 2 relay services, and 2 builders. The connection with the red cross will be restricted.

- **We set different transaction destinations for builder1 and builder2**: So that we can evaluate which of the two builder's blocks gets proposed by the victim, we set the destination of the transfer transaction to be two different wallets, as opposed to a random address. We set transactions populated at builder1 to have the destination address 0x11100.....000 and for builder2 the destination address 0x22200....000.

We define the following attack and theorize its outcome:

- **Scenario 2, Attack 1**: Drop a percentage of OUTGOING packets from relay1 to the mev-boost service.We theorize that the victim proposer does not receive the block submitted by relay1 and chooses the less profitable block submitted by relay2.

We again simulate 100 blocks for each level of traffic control and record the transactions included in these 100 blocks through the Victim's consensus client interface.

Definition 3.2 (Attack success rate)**.** We define the attack success rate as the ratio between successes and failures, where an attack is considered successful if and only if all the transactions recorded in the block have destination address 0x22200...000

3.4 Practical issues

During the setup of the development network, there were several practical issues that delayed or even hindered the procedure. The main cause was the lack of documentation on the subject of custom Ethereum devnets setup, with the most referenced tutorial having been clearly outdated and even misleading. We include therefore a number of tips that we have found useful on the setup of our custom Ethereum POS devnet in the appendix below for anyone looking to run their tests in the future.

Chapter 4

Results

4.1 Scenario 1: Network of one Relay and one Builder

4.1.1 Scenario 1, Attack 1

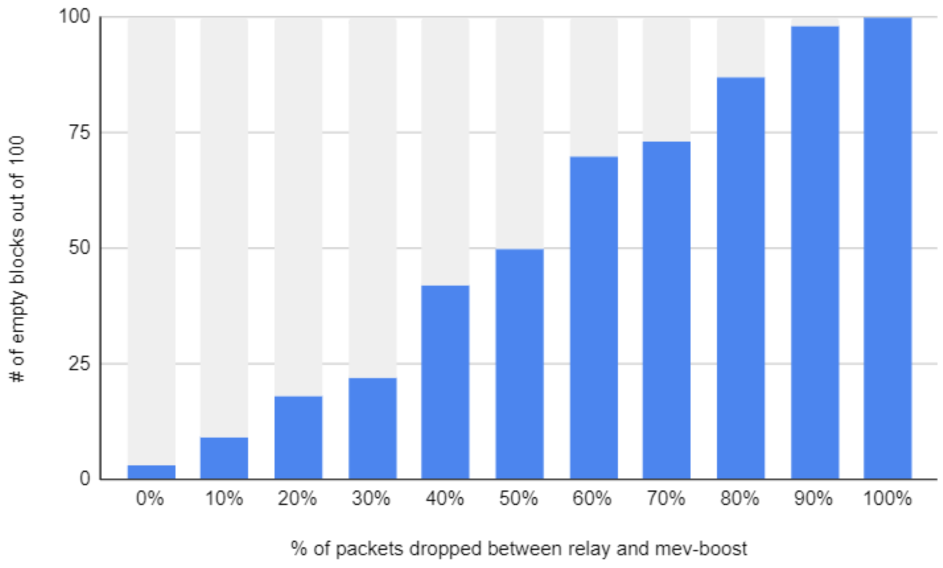


Figure 4.1: Attack success rate of Scenario 1, Attack 1. Blue: Number of empty blocks.

Figure 4.1 showcases the results of this simulation. We recognize first of all that there is some noise involved in the simulation. On a default environment with no traffic control in place, 3 of the 100 blocks were recorded empty. Verifying our logs, we were able to track down the issue to instances of "validator not found" on the builder's execution client. This is a failure on the routined query of validator duties, where it does not find a validator duty mapped to this block, and fails to build the block where the address of the validator on duty is needed.

Since an epoch consists of 32 slots, this matches the frequency of this error. We treat this discrepancy as noise and consider this through the rest of the analysis.

We see an exponential increase in attack success rate starting as early as the 10% drop in all outgoing packets, reaching a 100% success rate at 100% packets drop. TCP packets that do not reach their destination are generally re-transmitted until their timeouts reach. We assume that an attack succeeds if an individual packet does not reach its destination on time, even after being re-transmitted multiple times. Assuming a packet is transmitted in total x times and a packet drop rate of y, the probability of an individual packet not reaching its destination is

$$y^x$$

Assuming n unique packets are transmitted, the probability of at least one unique packet not reaching its destination, meaning the attack is successful, is

$$1 - (1 - y^x)^n$$

The attack success rate increasing exponentially with respect to the packet drop rate in practice falls in line with our analysis.

4.1.2 Scenario 1, Attack 2

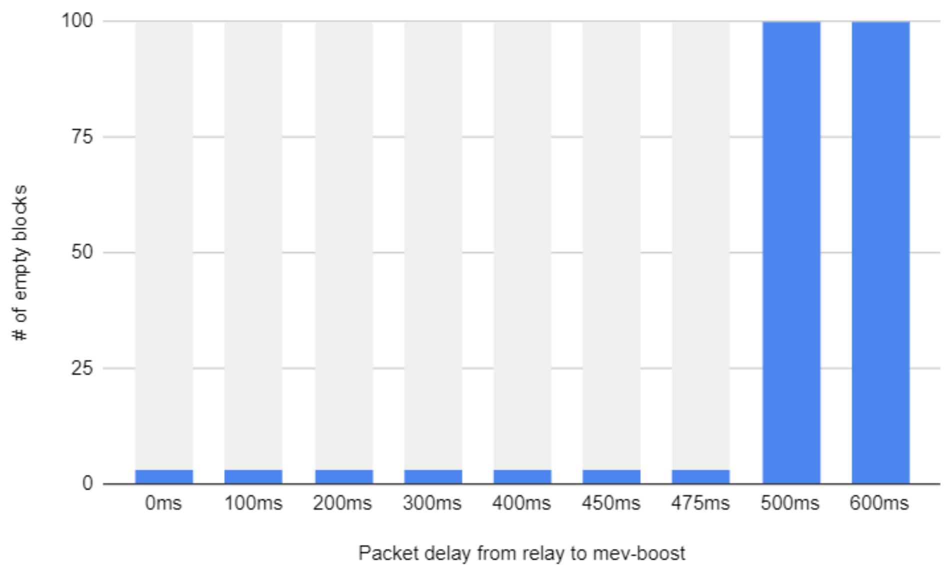


Figure 4.2: Attack success rate of Scenario 1, Attack 2. Blue: number of empty blocks

Figure 4.2 showcases the results of this simulation. We recognize that the same noise is present as in scenario 1 (see figure 4.1) and take this into account throughout the analysis. We observe a clear cut-off point of 500ms, below which the traffic control does not affect the blocks and above which the success rate reaches 100% instantly. This aligns with the TCP timeout offsets defined in the relay configurations, thus giving us a clear conclusion that the attack is effective on a delay above this timeout value.

4.2 Scenario 2: Network of two Relays and two Builders

4.2.1 Scenario 2, Attack 1

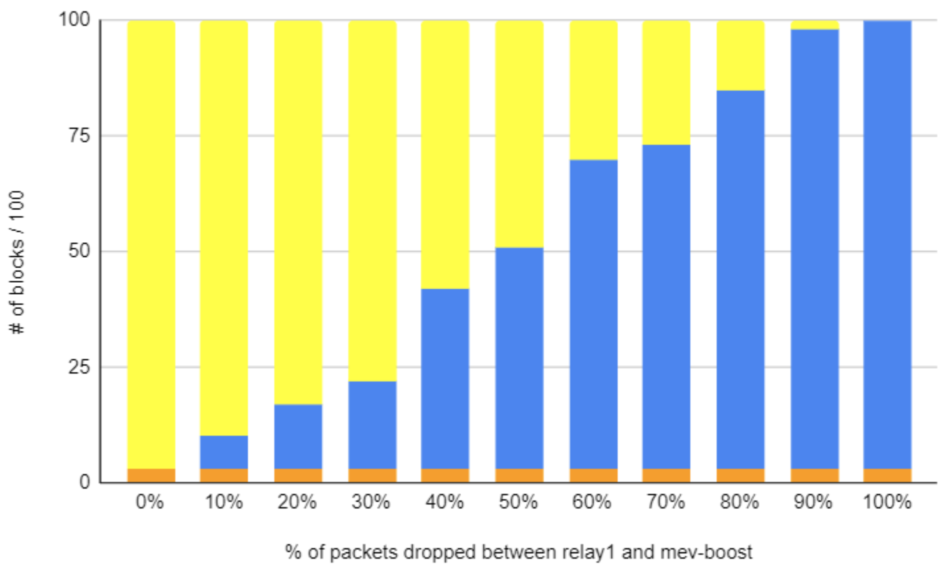


Figure 4.3: Attack success rate of Scenario 2, Attack 1. Orange: Empty blocks (noise). Yellow: Blocks from relay1. Blue: blocks from relay2.

Figure 4.3 showcases the results of this simulation. We recognize again that the same noise is present as in scenario 1 (figure 4.1) and take this into account throughout the analysis. We observe that the results are very similar to Scenario 1 Attack 1 (figure 4.1), we see an exponential increase in the amount of blocks included from relay2 as opposed to relay1 with respect to increasing drop rates. The difference in this scenario is that there exists a second relay submitting less profitable blocks than the first relay, to the mev-boost service, which is programmed to choose the most profitable block. We confirm in this simulation that this job is fulfilled in that the proposed blocks are not completely empty when the attack drops the packets sent from the first relay, but instead contain the blocks submitted by the second relay.

Chapter 5

Conclusion/Outlook

Based on the results of our attack simulations in chapter 4, we can conclude that it is indeed very possible to influence the outcome of the blockchain by controlling the traffic of the connection between relays and validators. Since these relays are publicly accessible servers and since there are currently only a select few trusted relays [2], the risk of such attacks is very much present on a centralized network landscape. Each entity involved in the PBS scheme can potentially be a malicious actor:

- Malicious builder: A builder that records a high-value transaction in their transaction pool, can try to restrict relay access to other builders in order to increase the chance of their block being selected and them receiving the transaction rewards.
- Malicious relay: A relay can see the contents of builder-submitted blocks and which block to relay. The more trusted it is, the more power it holds over the submitted blocks, as it can relay a different block to a validator than the actual highest-bidded one.
- Malicious proposer: A proposer may block another proposer's access to a specific relay. Say a proposer knows that it is on future duty within this epoch and that it records blocks of high value getting proposed due to high on-chain transaction volume, it can try to restrict relay access to validators on current proposal duty and get access to these high value blocks itself, earning it higher proposal rewards.

Bibliography

[1] Ethereum. Ethereum roadmap: Pbs. [Online]. Available: https://ethereum.org/nl/roadmap/pbs/

[2] L. H. et al., "Ethereum's proposer-builder separation: Promises and realities," in *arXiv preprint arXiv:2305.19037*, 2023.

[3] ycharts.com. Ethereum marketcap. [Online]. Available: https://ycharts.com/indicators/ethereum_market_cap

[4] Ethereum. Consensus specs. [Online]. Available: https://github.com/ethereum/consensus-specs/tree/dev/specs

[5] R. M. Karp, "Computers and intractability: A guide to the theory of np-completeness. w.h. freeman," 1972.

[6] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks*, vol. 44, no. 5, pp. 643–666, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128603004250

[7] Flashbots. Running-mev-boost-relay-at-scale. [Online]. Available: https://flashbots.notion.site/Running-MEV-Boost-Relay-at-scale-4040ccd5186c425d9a860cbb29bbfe09#5e5ae06bb7f2467796d73293ba2de5e8

[8] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on Bitcoin's Peer-to-Peer network," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[9] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[10] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 375–392.

[11] Ethereum. Client diversity. [Online]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/client-diversity/

[12] F. research org. Mev-boost. [Online]. Available: https://github.com/flashbots/mev-boost

[13] F. research org. Mev-boost relay. [Online]. Available: https://github.com/flashbots/mev-boost-relay

[14] F. research org. Flashbots builer. [Online]. Available: https://github.com/flashbots/builder

[15] Prysmaticlabs. Prysm consensus/validator clients. [Online]. Available: https://github.com/prysmaticlabs/prysm

[16] Ethereum. go-ethereum. [Online]. Available: https://github.com/ethereum/go-ethereum

[17] Docker. Docker website. [Online]. Available: https://www.docker.com/

[18] B. Hubert. linux-tc documentation page. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html#AUTHOR

[19] Ethereum. sse events repo. [Online]. Available: https://github.com/ethereum/beacon-APIs/blob/master/apis/eventstream/index.yaml

[20] Flashbots. mev-boost-relay/common/types.go. [Online]. Available: https://github.com/flashbots/mev-boost-relay/blob/98576112e245454a225e9a668cacfa498eea00b4/common/types.go#L117

[21] Flashbots. mev-boost-relay/services/api/service.go. [Online]. Available: https://github.com/flashbots/mev-boost-relay/blob/main/services/api/service.go

[22] Flashbots. mev-boost-relay/services/housekeeper/housekeeper.go. [Online]. Available: https://github.com/flashbots/mev-boost-relay/blob/main/services/housekeeper/housekeeper.go

Appendix A

Tips on setting up an PoS-PBS Ethereum Devnet using Docker

The implementation of this work has stemmed from the tutorial from the prysm documentation, which is now very much outdated. In this appendix, we lay out some tips that were useful to adapting to the outdated tutorial, in hopes of ever becoming useful for future simulations of devnets. The complete implentation can be found on the disco-students repo

A.1 Genesis nodes

The setup of the initial nodes of the network requires the generation of genesis configuration files. These can be done directly via the prysm consensus client and the geth execution client. During this, one must take into account that if one includes a list of validator clients in this manner, one must take note of the hash of these validators combined to make up the "genesis_validators_root" which is not immediately used but is an important configuration variable that is required by the relay API, as well as the Flashbots/builder.

A.2 Peer-to-Peer communication software version

It is not documented that p2p communication software used between versions of geth are different between geth versions. Specifically, at the time of writing, geth's latest Docker image on the docker hub (ethereum/client-go:latest) does not match the p2p software used on Flashbots' builders (flashbots/builder:latest). If an error log of instance "Remote needs update" or "Origin needs update" is present, these refer to the p2p software in use and signify that the nodes trying to connect to each other are not matching the software version. In this case, one must look in the log history for this software version and adapt the node version.

Our implementation circumvents this issue by using either one or the other for the entire node-set on the network.

A.3 Chain Checkpoints and node syncing

When a prysm-geth pair connects to an existing network (or the genesis node pair), one must wait until the finalization of the beacon chain headers for the prysm to become able to query the checkpoint-sync-url. One can check the status of the finalization by querying ".../eth/v1/beacon/headers/finalized" on the consensus node's grpc-gateway-port. Additionally, for a prysm-geth pair to sync to an existing network and participate as a network member, BOTH nodes must be in sync with the network, otherwise, certain endpoints or RPC calls will not function properly.

A.4 URIs

One must not forget to specify the protocol (i.e. "http://") when defining an endpoint in configuration files. Flashbots/builder's "–builder.beacon_endpoints" for example will not function if this prefix is forgotten. This is not specified in the documentation.

A.5 Relay Execution Client configuration

If Flashbots/builder is used as the blocksim execution client for a relay, one MUST set "–builder.dry-run=true" and "–builder.local_relay" AND additionally set the "–builder.remote_relay_endpoint" value for it to function as intended. The dry-run setting can be interpreted as the simulated block is not intended for immediate broadcast (as it is passed to the validator as for signing). This as well is not documented properly.

A.6 Relay API, housekeeper, and website services containerization

The relay's API, housekeeper, and website services are documented to be run through the terminal of a set server, however, these can be docker-containerized. In the case where this is done, one must not forget to set the environmental variables defined in the code (but not in the documentation) [20], [21], [22] to run these services. Otherwise, the services will not function as intended.

A.7 Traffic control environment

To enable the use of linux-tc, the container that is intended to run the traffic control requires the NET_ADMIN flag in docker compose. This flag enables administrative control over the docker network this container finds itself in.