Stress-Testing Ethereum Clients: Effects of Gas Griefing Strategies on Throughput and System Resources across Node Implementations

Expose – Second Draft

Trey Mosby

Goethe-Universität Frankfurt
FB 12 — Informatik und Mathematik
Bachelor Informatik

Veranstaltung:
Bachelor - Theses

Eingereicht von: Trey Mosby
7104949
Treymsby@gmail.com
s5487971@stud.uni-frankfurt.de

Studienrichtung: 12
10. Fachsemester

Abgabetermin: N/A

1 Contents

1 Contents **1**

2 Research Question **1**

3 Terms and Definitions **1**

4 Relevance **2**

5 State of research **2**

6 Methodic **4**

6.1 Experimental Design 4
6.1.1 Setup Phase 4
6.1.2 Role of Consensus Clients 5
6.1.3 Mixed Node Setup (Optional Extension) 6
6.1.4 Testing Phase 6
6.1.5 Evaluation Phase 6

7 Timeline **7**

8 Literature & Sources **7**

2 Research Question

I want to investigate the following Research Question:

How do different gas griefing strategies and attack vectors affect transaction times, resource utilization, and client performance across Ethereum node implementations in a controlled environment?

3 Terms and Definitions

- **Blockchain**: A decentralized, immutable ledger that stores transactions in cryptographically linked blocks and is managed via a peer-to-peer network [1], [2].
- **Ethereum:** A decentralized, open blockchain platform that enables the execution of smart contracts using the cryptocurrency Ether as an incentive and means of payment [1], [2].
- **Ethereum Virtual Machine (EVM):** A runtime environment that executes smart contract bytecode in a deterministic, stack-based virtual machine model, managing the state of the Ethereum blockchain via transition functions [1], [2]. It enables OP code instructions to be executed on nodes. [2], [3].

- **EVM OP code:** Machine-readable instructions in the Ethereum Virtual Machine that define and execute a specific operation within a smart contract bytecode [2], [4].
- **Gas:** Refers to the internal unit that measures the computing effort required for operations in the Ethereum Virtual Machine and acts as a fee mechanism [6].
- **Gas refund:** Partial refund of gas already used when memory is cleared [2], [5].
- **Proof of Work:** A consensus mechanism in which network participants ("miners") generate new blocks by solving cryptographic tasks and are rewarded with cryptocurrency [2].
- **Proof of Stake: A** consensus mechanism in which block generation and validation are proportional to the stake of cryptocurrency deposited in the network [2].
- **Gas griefing:** is a denial of service (DoS) attack strategy in which an attacker deliberately exploits or manipulates the gas costs of transactions to generate disproportionately high execution costs for legitimate users and thus disrupt operations [5], [7].

4 Relevance

Gas griefing can impair the stability and performance of Ethereum clients by exhausting block gas limits and overloading node resources (CPU, memory, I/O). Despite adjustments such as EIP-3529: Reduction in refunds, refund mechanisms that enable economically efficient abuse still exist.

5 State of research

The current state of research on gas griefing and related DoS attacks includes the following contributions:

1. **Fundamental works & protocol-side countermeasures**
    - *The Ethereum White Paper* describes the original design of gas costs and mechanisms for smart contracts [1].
    - *The Ethereum Yellow Paper* fundamentally defines gas costs and refund rules and forms the basis for all subsequent investigations [2].
    - **EIP-3529** (2021) reduces the maximum refundable gas amounts across the board as a direct response to early gas griefing cases, thereby reducing the economic incentive for these attacks [7].
    - **EIP-3298** (2021) introduced another measure that automatically catches insufficient gas specifications in smart contracts [9].
    - **EIP-7825** (2024, Nov) Sets a maximum limit of **16,777,216 gas** (2^{24}) per transaction, preventing single transactions from consuming an entire block's gas and thus reducing DoS attack potential. [15]

- **EIP-7791** (2024, Nov) Introduces a new opcode, `GAS2ETH`, to convert gas to ETH. Could introduce **new griefing vectors**. [16]
- **EIP-7904** (2025, May) Overhauls gas cost schedules for opcodes, precompiles, memory expansion, and data access according to actual computational complexity—seeking a fairer cost model. [17]

2. **Empirical case studies, systematic reviews & detection tools**
    - The first systematic framework is MadMax, which statically analyzes smart contract bytecode and identifies insufficient gas griefing as a separate attack type [10].
    - **Aghaei et al. (2024)** conduct a systematic review of security vulnerabilities in Ethereum smart contracts, which also comprehensively analyzes gas griefing attacks, their predictors, and countermeasures [13].
    - **Ghaleb et al. (2022)** present *eTainter*, a static analysis tool that detects gas-related vulnerabilities in smart contracts [14].
    - **Luo et al. (2018)** proposes an adaptive gas cost model that adjusts opcode-specific prices based on current network utilization to prevent underpriced DoS attacks [12].
3. **Further resources**
    - **Ethereum Wiki** provides continuously maintained documentation on gas, fees, and attack scenarios in the Ethereum ecosystem [3].

Research on practical effectiveness and countermeasures is fragmented. There is a lack of comparative experiments across clients and EVM implementations. Furthermore, most of the work was written in the proof-of-work era; there is a need for reevaluation and perspective expansion in the context of the implementation of proof-of-stake that has since taken place.

6 Methodic

I will investigate the effects of various gas griefing strategies on transaction time, resource utilization, and client performance across different Ethereum node implementations. A controlled, private Ethereum network will be constructed to systematically measure these effects.

6.1 Experimental Design

The experiment was divided into three phases: setup, testing, and evaluation. Each phase was structured to isolate the impact gas griefing under controlled conditions and enable comparative analysis between Ethereum clients.

Experiment Goals
The primary objectives were to:

1. The impact of gas griefing on transaction confirmation times and mempool behavior
2. The strain such attacks place on node-level system resources (CPU, memory, disk, and network I/O, Transaction volume, Transaction delay (latency/throughput), Transaction amounts (ETH/value per tx), Gas cost dynamics (Cost for honest User), Block utilization (blocks filled with junk txs, limiting space for legitimite activity)
3. Compare how different Ethereum clients (e.g., Geth, Besu) respond under stress.
4. Compare system performance when varying the number of nodes and the network topology.

6.1.1 Setup Phase

To begin, I'm establishing a controlled private Ethereum Test environment composed of local based nodes. This network will include at least 7 interconnected nodes (depending on available computational resources). I'll test multiple network topologies, and each deployment will be homogeneous. Every node in each setup will run the same Ethereum client, rather than mixing different clients.

Each node is set up in a container and will operate independently with its own configuration, while sharing a common file for consensus. Mining will be active to support autonomous block production.

Test accounts will be initialized through the block and assigned specific roles:

- **Griefer** to produce high-frequency, high-gas spam transactions.
- **Victim** to generate typical low-gas transactions.

Custom scripts will automate transactions several transaction types and track network behavior for consistent data collection.

Potential gas griefing strategies that can be used:

Technique	Category	Tactic	Impact
Opcode Gas Bombing	Opcode-level	Use expensive opcodes (e.g., SSTORE, CALL, BALANCE) in loops	Fills block gas limit; wastes miner resources
High-Gas Spam Tx Flooding	Mempool Saturation	Send many high-gas txs to self or dummies	Evicts legit txs; drives up network gas prices
Replace-by-Fee (RBF) Loop	Tx Replacement	Replace the same tx repeatedly with higher gas	Displaces victim txs; creates mempool churn
Nonce Griefing	Tx Replacement	Submit txs with target account's nonce range	Freezes target's future transactions
Block Saturation with Junk Tx	Gas Limit Exploitation	Fill blocks with many low-value txs that consume high gas	Leaves no space for useful txs
Contract Gas Baiting	Application-Level	Trigger expensive code paths in shared contracts	Raises execution cost for others

6.1.2 Role of Consensus Clients

In this setup, it is important to highlight the role of consensus clients. Ethereum's architecture after changing to Proof of Stake is split into two layers:

- **Execution Layer (EL)**, handled by execution clients such as Geth or Besu, which process transactions, execute smart contracts, and maintain the state [18].
- **Consensus Layer (CL)**, handled by consensus clients such as Lighthouse, Prysm, Teku, or Nimbus, which implement the Proof of Stake protocol, validate blocks and attestations, and ensure that all nodes agree on a canonical chain [18].

Consensus clients are necessary because execution clients alone cannot agree on the correct chain or finalize blocks. Without them, there would be no secure way to determine the legitimate history of transactions. By separating execution and consensus into distinct clients, Ethereum improves security, client diversity, and modularity [18].

Although this experiment primarily evaluates gas griefing effects on execution-layer performance, understanding the role of consensus clients is essential, since they maintain the underlying agreement that allows the execution clients' transaction processing to remain valid and trustworthy.

6.1.3 Mixed Node Setup (Optional Extension)

If time permits, an additional experimental phase will explore heterogeneous or mixed node setups. This would involve:

- Running nodes with different execution clients (e.g., Geth, Besu, Nethermind) within the same network.
- Combining different consensus clients (e.g., Prysm, Lighthouse, Teku) alongside execution clients.

Testing a mixed setup would provide insights into how Ethereum's emphasis on client diversity impacts security and performance in practice, and whether heterogeneous environments react differently compared to homogeneous deployments.

6.1.4 Testing Phase

This Phase involves two primary test scenarios:

1. **Control Test** — This focuses on standard transaction flow with no attack transations, serving as a benchmark for normal behavior.
2. **Griefing Test** — In this round, the griefer floods the network with spam to disrupt transaction inclusion, while the victim continues submitting transactions.

Throughout both tests, system metrics will be tracked in real time, including:

- Processor load, memory use, disk I/O, and network throughput
- Mempool changes and inconsistencies between nodes
- Transations times

A variation of the test will include swapping a Geth node for Besu Nodes to compare resilience under identical network stress.

6.1.5 Evaluation Phase

Once the testing is complete, I'll extract and organize the mentioned data. I plan to compare the control and griefing scenarios to assess the attack's impact and evaluate client-specific resilience based on performance metrics and system stress. Finally, I will document my findings using logs, graphs, and screenshots, and archive all data to support reproducibility and further analysis.

7 Timeline

Week	Paper	Methodology
1	- Define paper structure - Review Literature - Write State of Art section - Define Metrics for Performance Testing - Describe Tools used (EVMs, Node, Clients etc..)	- Set up EVM nodes and environments - Set up Test Accounts
2	- Describe environment, network topology, etc... - Describe Automation & Griefing Srkipts, Controll	- Write Automation Skripts - Write Griefing Skripts
3	- Write Methodic section - Background / Related Work section	Buffer: - Write Automation Skripts - Write Griefing Skripts New Tasks - Run Baseline (Control) Test
4	- Write Abstract (Bulletpoints)	- Run full tests and collect data on Transaction-level, Node-level & Block-level - Compare to baseline test - Start Analyze collected data
5	- Write Results / Analysis Section - Write Discussion / Interpretation - Visualize Data	- Analyze collected data - Compare Results - Draw Conclusions
6	- Write Introduktion - Write potential defenses or mitigations section - Write Conclusion	Finish Documentation: - My Setup - Screenshots, logs, or graphs etc..
7	- Finish Intro - Finish Abstract - Review Everything	
8	- Review Again - Submit Theses	

8 Literature & Sources

[1] V. Buterin: *A Next-Generation Smart Contract and Decentralized Application Platform*. Ethereum Foundation White Paper, 2025. Online: https://ethereum.org/en/whitepaper/, Abrufdatum: 27. July 2025.

[2] G. Wood: *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Ethereum Foundation Yellow Paper, 2025. Online: https://ethereum.github.io/yellowpaper/paper.pdf, Abrufdatum: 27. July 2025.

[3] Ethereum Foundation: *Frontier Release Notes*. Ethereum Wiki, 2015. Online: https://github.com/ethereum/wiki/wiki/Frontier-Release, Abrufdatum: 27. July 2025.

[4] Ethereum Foundation: *Opcodes for the EVM*. Ethereum Developer Documentation, 2025. Online: https://ethereum.org/en/developers/docs/evm/opcodes/, Abrufdatum: 27. July 2025.

[5] V. Buterin: *EIP-150: Gas Cost Changes for IO-heavy Operations*. Ethereum Improvement Proposals, 2016. Online: https://eips.ethereum.org/EIPS/eip-150, Abrufdatum: 27. July 2025.

[6] Ethereum Foundation: *Gas and fees*. Ethereum Developer Documentation, Feb. 25, 2025. Online: https://ethereum.org/en/developers/docs/gas/, Abrufdatum: 27. July 2025.

[7] Ethereum Improvement Proposal EIP-3529: *Reduce refunds*, 2021. Online: https://eips.ethereum.org/EIPS/eip-3529, Abrufdatum: 27. July 2025.

[8] Ethereum Improvement Proposal EIP-7778: *Prevent Block Gas Limit Circumvention by Excluding Refunds*, 2024. Online: https://eips.ethereum.org/EIPS/eip-7778, Abrufdatum: 27. July 2025.

[9] Ethereum Improvement Proposal EIP-3298: *Insufficient Gas Griefing Mitigations*, 2021. Online: https://eips.ethereum.org/EIPS/eip-3298, Abrufdatum: 27. July 2025.

[10] M. Nikolic et al.: *MadMax: Surviving Out-of-Gas Conditions in Ethereum Smart Contracts*. ACM CCS Workshop on Blockchain, 2018. Online: https://cs.pomona.edu/~michael/courses/csci190s21/papers/madmax.pdf, Abrufdatum: 27. July 2025.

[11] Y. Zhou, Z. Wang, X. Li: *Analysis of DoS by Gas in Ethereum Smart Contracts*. Proc. Blockchain Security Workshop, 2022. Online: https://scispace.com/pdf/systematic-review-of-security-vulnerabilities-in-ethereum-6pl6pkx8.pdf, Abrufdatum: 27. July 2025.

[12] X. Luo, H. Wang, C. Chen: *An Adaptive Gas Cost Mechanism for Ethereum to Defend Against Under-Priced DoS Attacks*. Technical Report, PolyU, 2018. Online: https://www4.comp.polyu.edu.hk/~csxluo/DoSEVM.pdf, Abrufdatum: 30. July 2025.

[13] J. Aghaei et al.: *Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract*. SciSpace, 2024. Online: https://scispace.com/pdf/systematic-review-of-security-vulnerabilities-in-ethereum-6pl6pkx8.pdf, Abrufdatum: 30. July 2025.

[14] A. Ghaleb, J. Rubin, K. Pattabiraman: *eTainter: Detecting Gas-Related Vulnerabilities in Smart Contracts*. In: Proc. of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Virtual, South Korea, July 18–22, 2022, pp. 728–739. DOI: 10.1145/3533767.3534378. Online: https://doi.org/10.1145/3533767.3534378 , Abrufdatum: 30. July 2025.

[15] Giulio Rebuffo, Toni Wahrstätte: *EIP-7825: Transaction Gas Limit Cap*. Introduce a protocol-level cap, 2024. Online: https://eips.ethereum.org/EIPS/eip-7825, Abrufdatum: 24. August 2025.

[16] Charles Cooper, Pascal Caversaccio: *EIP-7791: Introduces a new opcode, `GAS2ETH`, to convert gas to ETH*, 2024. Online: https://eips.ethereum.org/EIPS/eip-7791 , Abrufdatum: 24. August 2025.

[17] Jacek Glen, Lukasz Glen: *EIP-7904: General Repricing*. Gas Cost Repricing to reflect computational complexity and transaction throughput increase, 2024. Online: https://eips.ethereum.org/EIPS/eip-7904, Abrufdatum: 24. August 2025.

[18] Ethereum Foundation: *Konsens-Clients*. Ethereum Developer Documentation, 2025. Online: https://ethereum.org/de/developers/docs/nodes-and-clients/#consensus-clients, Abrufdatum: 24. August 2025.