DETER: Denial of Ethereum Txpool sERvices

Kai Li
kli111@syr.edu
Syracuse University
Syracuse, NY, USA

Yibo Wang
ywang349@syr.edu
Syracuse University
Syracuse, NY, USA

Yuzhe Tang ✉
ytang100@syr.edu
Syracuse University
Syracuse, NY, USA

ABSTRACT

On an Ethereum node, `txpool` (a.k.a. mempool) is a buffer storing unconfirmed transactions and controls what downstream services can see, such as mining and transaction propagation. This work presents thefi rst security study on Ethereum `txpool` designs.

We discoverfl awed transaction handling in all known Ethereum clients (e.g., Geth), and by exploiting it, design a series of low-cost denial-of-service attacks named DETER. A DETER attacker can disable a remote Ethereum node's `txpool` and deny the critical downstream services in mining, transaction propagation, Gas station, etc. By design, DETER attacks incur zero or low Ether cost. The attack can be amplified to cause global disruption to an Ethereum network by targeting centralized network services there (e.g., mining pools and transaction relay services). By evaluating local nodes, we verify the effectiveness and low cost of DETER attacks on all known Ethereum clients and in major testnets.

We design non-trivial measurement methods against blackbox mainnet nodes and conduct light probes to confirm that popular mainnet services are exploitable under DETER attacks.

We propose mitigation schemes that reduce a DETER attack's success rate down to zero while preserving the miners' revenue.

CCS CONCEPTS

• **Security and privacy** → **Denial-of-service attacks**; *Software security engineering*; *Distributed systems security*; • **Networks** → *Peer-to-peer protocols*;

KEYWORDS

Blockchains; Ethereum; Mempool/Txpool; Designfl aws; Unconfirmed transactions;

ACM Reference Format:
Kai Li, Yibo Wang, and Yuzhe Tang. 2021. DETER: Denial of Ethereum Txpool sERvices. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 23 pages. https://doi.org/10.1145/3460120.3485369

* ✉ Yuzhe Tang is the corresponding author.

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3485369

1 INTRODUCTION

Today, operational blockchains have grown into complex ecosystems providing a wide variety of services to decentralized application (DApp) users, such as mining pools, transaction relay, DApp-specific services (e.g., Gas station in Ethereum), etc. Denying these services is a real threat (e.g., the Nov. 2020 incident that disrupts a popular Ethereum relay service [10], denial of services/DoS attacks among Bitcoin mining pools [45, 48, 64] and Bitcoin spam campaign [38]). Such a threat is of interest to some actual blockchain participants, for instance, a service provider competing with the victim service for the same customer base, or a DApp user racing to win an auction over peer users.

Related works: In the existing literature, blockchain DoS security has been examined at different system layers, including P2P networks [37, 47, 53, 62], mining-based consensus [6, 55], transaction processing [19, 24, 38, 60], and application-level extensions such as smart contracts [26, 40, 57] and DApp (decentralized application) services [52]. Despite the extensive research, most existing works consider powerful institutional attackers, such as the ones able to control a significant portion (e.g., 51% or 21%) of computing power in a large blockchain [55], or the ones who can corrupt the underlying network infrastructure like ISP insiders [37, 62], or the botnet which has tens of thousands of IP addresses at her disposal for an eclipse attack [47], or a Bitcoin mogul willing to spend tens of thousands-USD worth of Bitcoin to launch a spam campaign [38]. Recently proposed are a class of low-cost attacks exploiting miner extractable value (MEV) in which the attacker send crafted transactions to front-run other transactions [42, 58, 61] and/or to bribe miners [50, 63, 65]. This class of attacks assume rational miners and have limited impacts (on few targeted victim transactions, instead of all transactions submitted in a period). The full related works are in Appendix 10. It is an open research problem whether an average user can mount a low-cost attack to disable a large-scale blockchain.

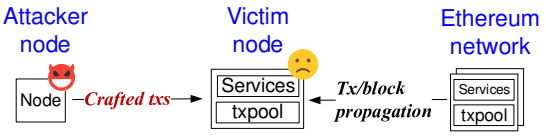


Figure 1: Threat Model: DETER a remote `txpool`

Attack goals: This work aims at low-cost DoS attacks targeting few blockchain nodes to cause large-scale impacts to the Ethereum-DApp ecosystem. The observation is that despite the original design to decentralize control, practical blockchain services, notably mining pools and transaction relay services, are highly centralized [46], a potential single point of failure. For instance, Ethereum blockchain's connection to millions of its active DApp users is controlled by a single relay service (infura.io [9]), whose outage in Nov.

2020 [10] (allegedly triggered by a competing service provider [32]) prompts major exchange platforms to halt Ether withdrawal, leading to a global panic among Ether holders. We thus consider the threat model depicted as in Figure 1 where an attacker node sends crafted transactions to a victim node running in a DApp service's backend (e.g., a mining pool or a relay service) which propagates transactions and blocks with the rest of Ethereum network. The attacker's goal is to disable the victim node's service to other nodes and DApp users at *a low Ether cost.* A baseline design is for the attacker account to send a large volume of transactions to occupy the limited block space and to squeeze out normal transactions there. However, the spam transactions sent in this baseline have to be with high Gas prices, burdening the attacker with high cost (e.g., 12.5 Ether per block as analyzed in § 3). Also, this baseline benefits miners by actually increasing their revenue, while this work aim to victimize the miners. Note that the attack goal in this work is to prevent a downstream miner from accessing the content of a `txpool`, which is distinct from the goal of MEV-exploiting attacks [42, 50, 58, 61, 63, 65] where the miner can access a `txpool` content and is incentivized to misbehave on it. Also, MEV-exploiting attacks make assumption on rational miners, which is not necessary for this work which aims to victimize any miners.

Attack design: To achieve the threat goals, this work exploits an understudied subject in the existing security literature, that is, denying the `txpool` service. On an Ethereum node, `txpool`[1] is an essential data structure that buffers unconfirmed transactions received from other nodes until they are included in the blockchain by miners. A disabled `txpool` presenting a false empty view of none transactions can cripple the victim node's mining and transaction propagation, and further lead to a global service disruption as will be confirmed by our evaluation.

To disable a `txpool`, our observation is the following: An Ethereum client's decision in whether to admit an incoming transaction affects the revenue of the miner running the client. Thus, any Ethereum client developed to attract revenue-hungry miners and increase adoption needs to maximize miners' revenue. To do so, real Ethereum clients are designed to take loose but risky actions to admit transactions, as we measured. The key idea of this work is to exploit real Ethereum clients' risky `txpool` behavior in admission control and to design denial-of-service attacks at low Ether cost, namely Denial of Ethereum's Txpool sERvice (DETER attacks). More specifically, one can trick a `txpool` to admit invalid transactions at zero Ether cost that evict and victimize existing normal transactions there. In the following, we describe two types of transactions that we exploit to design DETER attacks.

A future transaction in Ethereum (which is similar to the concept of orphan transaction in Bitcoin terms) is the one that arrives at a node earlier than its logical predecessor (i.e., the transaction of smaller nonces as will be introduced in § 2). On a node, a future transaction at the time of arrival, say tx_2 with nonce $n+2$, can wind up two outcomes when the node's miner reads it from the `txpool`: X1) It remains a future transaction which is invalidated by the miner and whose transaction fee cannot be collected, or X2) it is transformed into a valid (or pending) transaction upon the arrival of a subsequent transaction tx_1 of nonce $n+1$. In the latter case, tx_2's fee can be collected by the miner. To increase miners' revenue, all Ethereum clients, including Geth (Go) [8], Parity (Rust) [13], Nethermind (.Net) [12], and Besu (Java) [3][2] as we measured, support admitting future transactions, in the (optimistic) hope that case X2) will occur along the way and their transaction fees are collectible. In this work, we propose the first attack, named DETER-X, that exploits Ethereum's falsely optimistic design by deliberately sending future transactions of case X1) and at high Gas price so that they are admitted to the `txpool` and evict normal transactions, yet without being charged due to transaction invalidity. This attack victimizes not only the senders of the normal transactions evicted (by tx_1) but also the miners who end up mining no transactions and get low or even zero revenue. Note that at the time of tx_2's arrival, which outcome, X1) or X2), will occur fully depends on the arrival of tx_1, which is unpredictable.

The second type of Ethereum transaction exploited is what we call latent invalid transactions, that is, the transactions that are valid and admitted at the time of arrival, but are transformed into invalid transactions later when the miner reads it. Consider a sequence of transactions, $tx_0, tx_1, tx_2, \ldots$, sent from the same account e and each of which spends account e's full balance. Similar to the previous attack, this transaction sequence can wind up with two likely outcomes: Z1) without interference of other transactions, this transaction sequence will be invalidated by the miner except for tx_0, as $tx_i(\forall i > 1)$ overdrafts e's balance. Z2) upon the arrival of subsequent transactions to refill e's account, $tx_i(\forall i > 1)$ may be transformed into valid transactions whose fees are collectible by miners. As we measured, all Ethereum clients deployed on the mainnet take the risk to admit latent overdraft transactions in their `txpool`. We thus propose the second attack, named DETER-Z, to exploit this risky design by sending overdraft transactions at high Gas price to evict normal transactions in the `txpool`, at low Ether cost. In particular, we propose DETER-Z variant optimized for Geth's admission control so that the latent invalid transactions do not only evict transactions but also occupy the `txpool` for an extended period of time. We also propose evasion strategies on other clients to bypass their limits on the transactions from the same account.

The DETER vulnerabilities are specific to several unique designs in the Ethereum blockchain, which may render their applicability beyond Ethereum limited. Concretely, the cause of DETER-X vulnerability can be attributed to Ethereum's supports of future transactions, which meet the real DApps' demands to send transactions *hastily*[3], as evidenced in measurement studies such as [66]. Likewise, the DETER-Z vulnerability has the root cause in Ethereum's account model in which account balances can be arbitrarily updated, which is necessary to support smart contracts desirable by many DApps. By contrast, Bitcoin's UTXO model features limited updatability in which a transaction output can transition only from the unspent state (i.e., UTXO) to the spent, but not the other way around. The one-way updatability in the UTXO model renders Bitcoin immune to the DETER-Z attacks, as we investigated. Note that

[1] `txpool` in Ethereum is the same concept with the mempool in Bitcoin.

[2] In this paper, we discard the cases of Aleth (C++) [2] and Trinity (Python) [21] clients, because no mainnet nodes run the client and their code maintenance is discontinued (e.g., as of Apr. 2021 for Aleth).

[3] Sending a transaction, say tx_2, hastily means sending it without waiting for the confirmation of other transactions that tx_2 depends on. For instance, such a dependent transaction, say tx_1, can be such that tx_2's nonce is tx_1's nonce plus one.

the two features exploited by DETER-X/Z attacks, namely future transactions and updatable accounts, are generic in the Ethereum protocol and are independent of specific client implementations.

Measurements and impacts: We evaluate both DETER attacks' impacts systematically on an extensive list of victim services in different settings. First, we evaluate the effectiveness and cost of DETER attacks against four Ethereum clients in a local setting (Geth/Parity/Nethermind/Besu) that show DETER-X achieves a 100% success rate at zero cost on all four clients except for parity with a success rate of 75.6% and zero cost. DETER-Z is with 100% success rate against all four clients and has a cost as low as 0.021 Ether per block, a three-order-of-magnitude saving from the baseline [38]'s cost of 12.5 Ether per block (§ 5).

Second, we evaluate the DETER's effectiveness on disabling blockchain mining: Against a single miner, we propose attack strategies that guide the timing of sending DETER-X payload by predicting the block arrival time based on the Poisson model [49, 59]. The attack evaluation on a local network shows that mounting DETER-X/Z attacks at a rate of 4 crafted messages per second against a single miner node can persistently reduce the block size (i.e., total Gas of transactions included in a block) by 88.8%/99.2%, at a 0/0.021 Ether per block. In addition, we evaluate DETER's effectiveness on a number of `txpool`-based services beyond mining, which includes transaction propagation, Gas stations [29] and other DApp services. The results confirm DETER's effectiveness and low costs, as are summarized in § 13.5 and described in more details in Appendix 13.

Third, we measure the mainnet and testnets' exploitability under DETER attacks. Measuring the mainnet in the presence of testnet results is necessary as a network's exploitability, described next, is specific to the network's deployment. Given each network, we measure whether critical nodes can be discovered (node discoverability) and whether the discovered node can be successfully attacked (node exploitability). For node discoverability, we propose two complementary measurement methods to improve the result accuracy. Measuring node exploitability in a live network poses challenges, as 1) the target node is a blackbox to us except for a minimal information-propagation interface, and 2) the measurement is heavily restricted by ethical concerns, esp. in the mainnet. We tackle these challenges and propose lightweight probe tests by exploiting Ethereum's support of transaction replacement. The key idea is that a node's internal state, such as a transaction's presence in the `txpool`, can be detected by the success of an attempt to replace this transaction with the one at a carefully selected Gas price; the success/failure of such an attempt is observable.

Our exploitability study shows all the mainnet nodes we discover are vulnerable under the proposed probes. The result on Ethereum's Ropsten [16] and Rinkeby [15] testnets shows that mounting DETER-X (DETER-Z) attacks on Rinkeby [15]'s top-5 miners can effectively reduce the testnet's number of transactions included in a block by $\frac{1}{77}\times$ (to zero) (§ 6.2).

Mitigation: DETER vulnerabilities are not by accident, and they are caused by the fundamental difficulty to design a "perfect" `txpool` in the tradeoff between DoS mitigation and miners' revenue on Ethereum. Specifically, mitigating DETER attacks perfectly without losing any miners' revenue is hard, because it requires such a `txpool` to look into the future just to make an admission decision at present. For instance, suppose an arriving transaction, say tx, is currently a future transaction. A perfect `txpool` needs to admit tx if the transaction will turn into a profitable pending transaction in the future, or otherwise, decline tx. However, whether tx would turn into a pending transaction depends on the arrival of subsequent transaction (e.g., another tx' from the same sender but with a smaller nonce), and thus is uncertain. With this uncertainty, a practical `txpool` may have to risk admitting a current pending transaction whose (high) fees may end up being non-collectible and/or declining a current future transaction whose (high) fees can become collectible in the end. The former decision is a DoS exploit while the latter one leads to loss of miner revenue.

Thus, in this work, we propose heuristics for `txpool` admission control. In the proposed schemes, we define necessary (but maybe insufficient) conditions to describe a DETER transaction and use them to detect/decline transactions. In other words, the scheme puts attack mitigation over miner revenue preservation. We evaluate the proposed mitigation schemes and show that 1) the mitigation schemes fail all DETER probe tests, resulting in zero attack success rates, and 2) the mitigation schemes well preserve and even increase miners' revenue under real Ethereum transaction traces. We conduct the cost evaluation by replaying the transaction traces collected from the mainnet.

Contributions of this work are listed as follows.

• *New attacks*: We discover Ethereum clients' vulnerability in managing unconfirmed transactions. We design two low-cost attacks, DETER-X/Z, to disable a remote Ethereum node's `txpool` service. We propose attack strategies targeting various node services in mining, transaction relay, and Gas station.

• *New understanding*: We measure and verify DETER-X/Z's effectiveness and low-cost extensively, in the settings of a local node running different Ethereum clients, testnets and the mainnet. We propose a non-trivial method to detect exploitability of a blackbox mainnet node by exploiting Ethereum's transaction replacement support. The results show DETER-X/Z vulnerability widely exist among Ethereum clients, a DETER attack can cause testnets (Rinkeby and Ropsten) to produce empty blocks at zero Ether cost, and mainnet nodes underneath critical services can be discovered and are tested exploitable.

• *Mitigation*: We propose mitigation schemes that can reduce the DETER attack success rate to zero while preserving the miners' revenue. We verify the properties of security and miners profitability under real and synthetic transactions.

Roadmap: § 2 introduces the background of Ethereum transaction processing and `txpool`. Threat model is presented in § 3. Two DETER attacks are described in § 4, and strategies to DETER the mining service are presented in § 5. § 6 presents the measurement study on the exploitability of real-world Ethereum networks under DETER attacks. Mitigation schemes are presented in § 7. Responsible disclosure is discussed in § 8 with the conclusion in § 9. In Appendices are full related works in Appendix 10, attack evaluation on services beyond mining in Appendix 13, among others.

2 PRELIMINARY

This section presents the background on Ethereum's transaction buffer, namely `txpool`, which is the foundation of understanding

DETER attacks. To begin with, we first describe the transaction workflow in Ethereum.

Transaction workflow: In Ethereum, the life cycle of a transaction begins from its owner account signing the transaction and sending the transaction to an Ethereum node, say N_1, typically through a remote-procedure call (RPC) interface[4]. Node N_1, receiving the transaction, conducts checks on transaction validity and priority (as will be elaborated on) before buffering it locally in a data structure called `txpool` and further propagating it to N_1's neighbors, say one of which is N_2. N_2 similarly propagates the transaction to its neighbors, and the process repeats until the transaction is propagated to the entire blockchain network. In the network, each miner node selects a group of unconfirmed transactions buffered from its `txpool` and runs mining algorithms on the selected group of transactions. Miners who find solutions of the mining puzzle prepare a block and propagates it to the network. A node receiving multiple blocks (at the same height) selects the first one and verifies its mining solution. If it passes, the node removes the transactions included in this block from its local `txpool`.

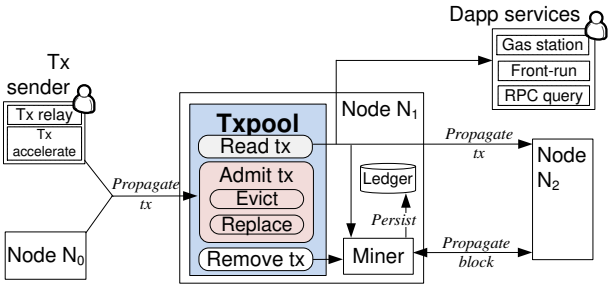


Figure 2: Ethereum transaction workflow and the role Node N_1's `txpool` plays in it.

Txpool operations: Figure 2 illustrates the workflow of transaction and block propagation described above. It takes a `txpool`-centric vantage point on Node N_1. The `txpool` supports three essential operations: 1) An incoming transaction, either directly sent from the sender account or propagated from a neighbor of N_1, is admitted into `txpool` (*transaction admission*). Admitting a transaction tx may trigger two `txpool` events: 1a) *eviction* of an existing transaction tx' by tx where tx and tx' are of different sender accounts or nonces, and 1b) *replacement* of an existing transaction tx' by tx where tx and tx' are of the same sender and nonce. 2) An existing transaction in the `txpool` is read by a downstream component (transaction read); for instance, a group of transactions in the `txpool` may be selected by a miner to include in the next block. Or transactions in the `txpool` may be read to determine the validity of an incoming transaction. Or unconfirmed transactions in `txpool` are read by DApp-specific (decentralized application) services such as Gas stations [7, 29] (to suggest cost-effective Gas prices in real time), DeFi frontrunning bots [43], RPC queries (e.g., the RPC API `txpool_content` [18]), etc. 3) Unconfirmed transactions in the `txpool` are emitted to the local miner upon finding or receiving a valid block of transactions.

Ethereum transactions: An Ethereum transaction binds a sender account to a receiver account, where an account is a public key of an Ether owner.[5] **Nonce**: Ethereum supports "hasty" transaction sending, that is, an account can send a transaction, say tx, without waiting for the confirmation of the transaction tx depends on. To enforce a total-order among hastily sent transactions, each Ethereum transaction is associated with a *nonce*, which records a monotonically increasing counter value per each sender account. In a `txpool`, a transaction is of state *pending*, if its nonce equals one plus the maximal nonce of the transactions of the same sender in the `txpool` (i.e., equal to $n+1$). Otherwise, if the nonce is strictly larger than $n+1$, the transaction is a *future* transaction. A future transaction can be a result of the Ethereum network propagating hasty transactions out-of-order. **Gas price**: In Ethereum, each transaction needs to specify a *Gas price*, that is, the amount of Ether the sender is willing to pay to a miner for each unit of "work" it does for including the transaction into the blockchain. Here, the work refers to the basic transaction validation workload and that for executing the smart contract invoked by the transaction. The work unit is Gas.

RPC services such as infura.io, etherscan.io, quiknode.io have been the primary means DApp clients use to communicate with the Ethereum blockchain. For instance, the transactions sent through infura.io alone accounts for at least 63% of all Ethereum transactions [25]. A RPC service receives from DApp clients JSON data on its frontend and sends a transaction packing the data to the blockchain on the backend. Specifically, the RPC service runs several Ethereum nodes on the backend that propagate transactions/blocks with the blockchain network.

Mining pools. Minimally, a mining pool mines a block by decomposing a puzzle into several easier versions, sending to miner participants decomposed puzzles and collecting their solutions (so-called shares) before paying out rewards to miners. Today, most blocks on Ethereum are mined by mining pools, and a typical mining pool is a complex service extended with two frontend capabilities: an own RPC service that directly accepts its customers' transactions and P2P service that propagates transactions and found blocks. For instance, Sparkpool, the most popular mining pool in Ethereum, has a RPC-and-P2P frontend service named Taichi network [33] through which users can receive/send transactions/blocks.

3 THREAT MODEL

The threat model consists of four actors in an Ethereum network. *An attacker* controls an external owned account (EOA) and a full node running Geth client that is connected to an Ethereum network, such as an Ethereum mainnet. A victim node running the Geth client contains *a victim txpool* and a series of *downstream services* that read `txpool` data. The last actor in the threat model is the rest of the *Ethereum P2P network*. The nodes in the Ethereum network run client software such as Geth and Parity, and they conform to the Ethereum's protocols for transaction/block propagation. The threat model is illustrated in Figure 1.

Particularly, we assume the attacker node has been connected to the victim node as its neighbor and can propagate messages of crafted transactions. The feasibility of this assumption is practically evaluated in both Ethereum testnets (§ 5) and mainnet (§ 6.3); Very briefly, an attacker can discover the victim node, learn the victim's

[4]We will use "RPC service" and "transaction relay", interchangeably in this paper.

[5]In this work, we don't exploit the smart-contract capability of Ethereum and only consider the basic transaction functionality for Ether transfer.

IP/port/nodeID via measuring the public Ethereum network, and connect to the victim node via passively waiting for the incoming connections from or via proactively initiating the handshakes to the victim node.

The attacker's goal is to deny the service of `txpool` to downstream components. More specifically, when the `txpool` is read by these components, a falsely empty snapshot of `txpool` is read and legitimate transactions are discarded from the view of the downstream services or other nodes. To do so, for instance, the attacker may *purge* for once all the transactions currently residing in the `txpool` and/or *occupy* the `txpool` with her own transactions to deny the service to other senders' transactions.

Via a few disabled `txpool`'es, the attacker wants to eventually victimize other accounts whose transactions cannot get propagated to the Ethereum network or included in the blockchain. Also, the attacker aims to victimize the miners and decrease their revenue by tricking them to mine on "empty" blocks. The attacker also aims to disable or manipulate DApp services that depend on unconfirmed transactions in `txpool`, such as manipulating the Gas price prediction by a real-time Gas station running on `txpool`.

Besides the attack effectiveness, the attacker's secondary goal is to lower the cost of her attack. Specifically, the cost of a DETER attack should be lower than the cost of the baseline attack (described next) by orders of magnitude.

Analyzing a baseline attack: A baseline attack works by the attacker account sending a flood of spam transactions to "frontrun" other normal transactions and to occupy the limited space of Ethereum blocks. The attacker can do so by configuring her transactions at a higher Gas price than the normal transactions and thus receiving a higher priority to be processed by miners (i.e., 1000 Gwei). To occupy one Ethereum block without any space for normal transactions, the baseline attacker needs to send spam transactions worth Gas of one block limit (i.e., 12.5 million Gas).

We observe the highest Gas price in the recent 200 blocks (from height 12202391 to 12202591) is consistently below 500 Gwei, based on the Gas station [31]. We use 1000 Gwei to estimate the highest Gas price in a block. Thus, the total cost of a baseline attack occupying a single block is $12.5 \cdot 10^6 \cdot 1000 = 12.5$ Ether. As of this writing (on Apr. 2021), this cost is equal to 24795 USD.

The baseline spam attack does not achieve the threat goal aimed in this work: First, the baseline attack incurs high monetary cost (12.5 Ether for one block). Second, the baseline attack, while victimizing normal transaction accounts, does not deny the service of Ethereum blockchain itself. Particularly, the miners under the baseline attack still receive high revenue, actually higher than normal due to the high Gas price of the spam transactions in the attack.

4 DETER ATTACKS AT A TXPOOL

In this section, we describe the attacks to disable a victim node's `txpool`. The next section describes the broader impacts of a disabled `txpool` to the downstream services on the victim node and other nodes in the Ethereum network.

We first describe design motivations based on the observation of profiling `txpool` in § 4.1. We then present two attack designs, DETER-X and DETER-Z, respectively in § 4.2 and § 4.3. We propose attack strategies to victimize a local miner in § 5 and evaluate the attacks against the local miner in § 5.1.

4.1 Observing txpool's Eviction Behavior

Table 1: Notations

Notation	Meaning
n	Max `txpool` length (e.g., default value 5120)
$L/M/H$	Symbolic value for high/median/low Gas price
$V/F/I$	Valid/future/invalid transactions

Our attack design exploits the eviction (mis)behavior in Ethereum clients' `txpool`. This subsection presents generic test cases against a remote blackbox `txpool` to characterize its eviction behavior.

`txpool` tests: In a `txpool` of pre-state S, an eviction takes as input an incoming transaction tx and produces as output a post-state S' and an eviction victim tx'. We denote an eviction event by state transition $(S, \{tx\}) \rightarrow S', \{tx'\}$. When no transaction is evicted in a full `txpool`, it could be $(S, \{tx\}) \rightarrow S, \varnothing$.

In our model, an Ethereum transaction tx is with two properties: the transaction state $[VIF]$ and Gas price $[HML]$. Transaction state can take values such as valid pending V, invalid pending I and future transactions F. The property of Gas price can take symbolic values such as high H, medium M and low L. For instance, VL denotes a valid pending translation with low Gas price, and an overdraft pending transaction at high Gas price is denoted by IH.

Given a `txpool` of capacity n (i.e., storing at most n transactions), we design two general tests:

t_1 Test $t_1(x_1 \cdot VL + (n - x_1) \cdot FM, FH) \rightarrow$ ✓|✗|✗. The initial state S contains x_1 valid pending transactions at low Gas price (VL) and $n - x_1$ future transactions at medium price (FM). The incoming transaction is a future transaction at a high Gas price FH. All transactions are sent from different accounts.
The partial success of the test, denoted by ✓, indicates that in the target `txpool`, a future transaction ($\{tx\} = \{FH\}$) can evict a valid pending transaction ($\{tx'\} = \{VL\}$). Additionally, if the evicting future transaction can persist and is stored in `txpool`, the test is a full success, denoted by ✓. Otherwise, that is, when there is no eviction or the eviction victim is FM, the test is a failure denoted by ✗.

t_2 Test $t_2(n \cdot VL, VH + (x_2 - 1) \cdot IH) \rightarrow$ ✓|✗|✗. The initial state S contains n valid pending transactions at low Gas price (VL). The incoming message $\{tx\}$ contains a valid transaction (VH) followed by $x_2 - 1$ overdraft transactions (IH) sent from the same account. These incoming transactions are at a high Gas price IH. The partial success of the test, denoted by ✓, indicates that the overdraft pending transaction ($\{tx\} = VH + (x_2 - 1) \cdot IH$) can evict valid pending transaction ($\{tx'\} = x_2 \cdot VL$). Additionally, if the evicting overdraft transactions of the same sender can persist in `txpool`, the test is a full success, denoted by ✓. Otherwise (i.e., no eviction by any of the $x_2 - 1$ overdraft transactions), the test is a failure denoted by ✗.

Test results of real Ethereum clients: In our study, we set up 1) a measurement node running instrumented Geth and the test code and 2) a target node running one of the following Ethereum clients: Geth (Go), OpenEthereum/Parity (Rust), Nethermind (.net), and Besu (Java). We statically instrument the test Geth node so that it can bypass local checks and propagate any transactions including future and invalid transactions to the target node. We consider the four clients that are deployed on Ethereum mainnet. The percentage

of nodes running these four clients on mainnet is illustrated in the second column of Table 2, from which Geth (83%) and Parity (15%) are the dominant clients on the mainnet.

Table 2: Profiling different Ethereum clients under tests and DETER attacks. For the two DETER attacks, success rates are reported. DETER-Z's Ether cost is also reported in parenthesis. Note that the baseline attack incurs a cost of 12.5 Ether per block. The second column refers to the percentage of mainnet nodes running a specific client [27]. Note the Ether cost is tested under a `txpool` storing transactions of the maximal Gas price 1000 Gwei.

Ethereum clients	Percentage	t_1	t_2	X	Z (Ether)
Geth	83.24%	✓ ($x_1 < 1024$)	✓ ($x_2 < 4096$)	100%	100% (0.021)
		✗ ($x_1 \geq 1024$)	✗ ($x_2 \geq 4096$)		
Parity	14.57%	✓ ($x_1 \geq 2000$)	✓ ($x_2 \leq 81$)	75.6%	100% (2.1)
		✗ ($x_1 \leq 2000$)	✗ ($x_2 > 81$)		
Nethermind	1.53%	✓	✓($x_2 \leq 17$)	100%	100% (1.28)
			✗ ($x_2 > 17$)		
Besu	0.52%	✓	✓	100%	100% (0.021)

We run the two tests against the four Ethereum clients and evaluate the success of each test. The result is presented in Table 2. **Test** t_1 fully succeed on all clients except for certain conditions under Parity and Geth: 1) When $x_1 \leq 2000$, Parity does not evict any pending transaction for an incoming future transaction, rendering a failed t_1. 2) On Geth, when $x_1 \geq 1024$, the future transactions do evict existing pending transactions but only 1024 transactions are admitted to the`txpool`, rendering a partially successful t_1. **Test** t_2 is successful on all clients excepts for the case that when x_2 is larger than 81 (17), t_2 fails on Parity (Nethermind). The failure of these test cases is due to that these clients enforce the limit of transactions from the same account. 3) When x_2 is larger than 4096, t_2 is partially successful on Geth. That is, sending $x_2 > 4096$ transactions as in t_2 to Geth, the `txpool` exhibits a behavior that evicts x_2 existing pending transactions and admits only 4096 transactions (the other $x_2 - 4096$ transactions do evict existing transactions but are not admitted to the `txpool` by themselves).

4.2 DETER-X: Exploit Future Transactions

Design motivation: The property tested in t_1 can be exploited to disable a `txpool`. That is, property t_1 implies that a `txpool` receiving a future transaction of high price chooses an existing pending transaction of low price to evict. One can abuse this property by sending a large number of future transactions to evict all pending transactions on a `txpool`. To evade certain clients' limit of transactions from the same account (recall test results t_2), the attacker prepares multiple accounts to send the future transactions. Because the future transactions, however high their Gas prices are, do not charge their sender, the attack that only sends future transactions incurs zero Ether cost.

Basic attack workflow: Formally, in our threat model, the attacker initially makes a guess about the value n' that is larger than the victim `txpool` length n, such as $n' = 10 \times 5120$ where 5120 is the default n in Geth.

x1: The attacker sends a crafted message encoding n' future transactions to the victim node. Each of these future transactions is configured with a very high Gas price, much higher than any existing transactions in `txpool` (e.g., a practical value is 1000 Gwei).

The success of the attack (DETER-X) is defined as that the downstream service on the victim node reads a falsely empty set of transactions from the `txpool`.

Attack analysis on Nethermind/Besu: Among these clients, a single step of x1 suffices to evict all pending and future transactions in the `txpool`. The crafted future transactions will stay in the `txpool`, preventing the transactions subsequently arriving at the `txpool` from being admitted. In other words, DETER-X by conducting a single step of x1 can *occupy* the `txpool` forever. Note that those crafted future transactions cannot be removed upon the miner finding the next blocks.

Attack customization on Geth: On Geth, a single step of x1 can evict all pending and future transactions in the `txpool`. However, after that, Geth limits the maximal number of future transactions that exist in `txpool` by $n_f < n$. Thus, only n_f (e.g., 1024) crafted transactions sent in x1 can persist in the `txpool`. Subsequently arriving transactions will be admitted to and repopulate the `txpool`.

Thus on Geth nodes, the attacker needs to periodically run step x1 at a certain frequency. Ideally, assume the attacker can know (or predict) the next time point, say T_i, when the `txpool` will be read by the local miner (e.g., upon finding a block) or another downstream service. Estimating the next T_i is specific to services and will be described in § 5.

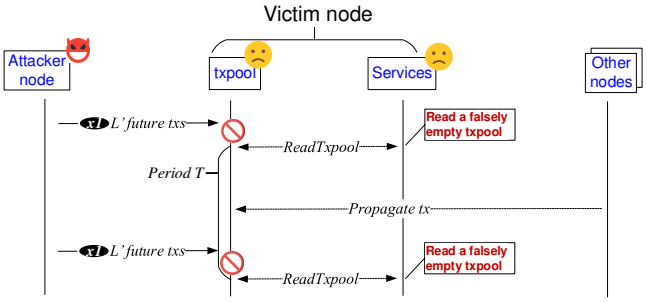


Figure 3: Workflow of a successful DETER-X attack that purges a victim `txpool` twice, so that the downstream service reads a falsely empty `txpool` twice.

Knowing T_i, the attacker sends her crafted future transaction in x1 right before T_i to reduce the number of subsequent transactions repopulating `txpool`. After this round, the attacker waits until the next estimated read time T_{i+1}. She then repeats the action of sending future transactions in step x1. The entire attack workflow is illustrated in Figure 3.

4.3 DETER-Z: Exploit Latent Invalid Transactions

We design adaptive attacks to exploit the behavior tested in t_2, that is, a `txpool` may admit invalid transactions by evicting valid transactions. Intuitively, an attacker can send a sufficient number of invalid transactions at high Gas price to evict existing pending transactions at a lower price, thus purging the `txpool`. As the invalid transactions don't charge their senders, this attack is of low cost. We thus propose to construct the basic DETER-Z attack:

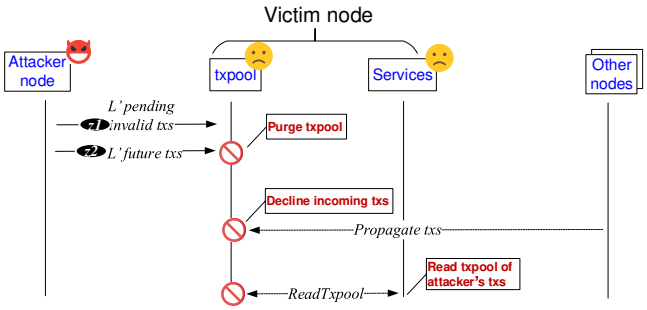


Figure 4: Workflow of a successful DETER-Z attack; note that step z1 applies to all Ethereum clients and step z2 is required only for Geth.

z1: In our threat model, the attacker node sends a message of n' pending transactions to the victim node. The pending transactions are sent from the same account of balance B, and each of them spends B Ether minus the transaction fee. The nonces of these transactions are consecutive. We call these transactions latent invalid transactions because each individual transaction is valid but together all n' transactions except for the first are overdraft transactions. The first transaction with the smallest nonce is denoted by tx_0. All n' transactions are with high Gas price, say pc. pc is slightly higher than any existing transaction in the `txpool`.

Attack analysis on Besu: Running z1 on Besu is effective, as Besu does not limit the number of transactions by the same sender.

Attack customization on Geth: Geth allows the eviction of existing pending transaction by latent overdraft transactions, but may not admit these overdraft transactions in the `txpool`. Running the step z1 would leave empty slots in the `txpool`, which allows the subsequent transactions to be admitted. To avoid this and to make the attack payload occupy the `txpool`, we propose the following attack workflow.

Against a target Geth node, the attacker first still runs Step z1 by sending n' transactions of the same account. She then the second step as follows.

z2: The attacker sends a message of n' future transactions to the victim Geth node. Each n_s future transactions of the n' ones are sent from the same account, so there are a total of $\frac{n'}{n_s}$ different accounts. The maximal Gas price of existing transactions in the `txpool` is 1000 Gwei, which is higher than 99.9% of transactions on the mainnet. The future transactions are configured with the Gas price higher than all existing transactions in the `txpool` but slightly lower than pc, say $pc - 1$. The two-step design of DETER-Z a Geth node is depicted in Figure 4.

Attack analysis on Geth: After the second step z2, the victim Geth node stores the maximal number of future transactions allowed by its `txpool`. As Gas price $pc - 1$ is lower than the pending transactions' price in z1, these future transactions would not evict the latent invalid transactions sent in z1.

After the two steps, consider that the victim node receives a legitimately propagated transaction tx'. As a normal transaction, tx''s Gas price would be much lower than pc. Thus, it is declined and would not be admitted into the `txpool`.

Now consider that a block that includes tx_0 arrives at the victim node or is found by the node itself. Receiving such a block, the node would evict tx_0 and all other pending transactions sent in Step z1. Thus, the `txpool` is fully occupied by the attack payload between z2 and when the next block is received.

Attack customization on Parity/Nethermind: These two clients limit the maximal number of transactions per sender in their `txpool` (by $n_s = 81$ and 17, respectively). To evade the limit, the attacker runs a customized Step z1 where she creates n'/n_s accounts, and under each account, sends n_s transactions of $VH + (n_s - 1) \cdot IH$. Using multiple accounts increases the cost of the attack.

4.4 Evaluation on a Local `txpool`

This subsection presents the evaluation to answer the following research question (RQ).

> **RQ1.** How effective and costly is a one-shot DETER attack in disabling a target `txpool`?

With the same setting (as described in § 4.1), we run the two DETER attacks. Initially, the target node is loaded with pending transactions to its capacity, that is, n_p transactions sent from n_p different accounts. During the attack, the test node sends a single round of DETER payload (i.e., Step x1 in DETER-X and z1/z2 in DETER-Z) to the target node running different clients. It then immediately reads the `txpool` content by issuing a `txpool_content` RPC request [18]. We calculate the number of evicted transactions by subtracting from n_p the number of original pending transactions found in the RPC result. The number of evicted transactions over n_p is reported as the attack's success rate. For instance, an attack of 90% success rate would evict 90% pending transactions that would otherwise be accessible by a downstream service. DETER-X has zero cost by design. For DETER-Z, we then turn on the target node's mining for long enough and observe which transactions are included in the blocks produced. We report the DETER-Z's cost by the Ether of the test node's transactions in all the blocks (i.e., a transaction's Ether is the product of its Gas and its Gas price).

The results are presented in Table 2. In general, the success rates of DETER-X and DETER-Z are 100% and DETER-Z's cost is 0.021 Ether per block, with the following notable exceptions: 1) DETER-X against Parity has a 75.6% success rate because Parity disallows the eviction of pending transactions by future transactions when there are fewer than 2000 pending transactions in the `txpool`. 2) DETER-Z against Parity incurs 2.1 Ether per block, because it limits up to 81 transactions per sender account. A DETER-Z attack has to prepare multiple accounts, each sending 81 transactions, in order to evict all pending transactions. 3) Similarly, DETER-Z against Nethermind incurs 1.28 Ether, because of the similar reason, that is, Nethermind limits a maximal 17 transactions of the same sender.

In summary, a DETER-X attack is always of zero cost (in both Gas and Ether), as future transactions don't charge fees. A DETER-Z attack's cost is due to the first transaction, which is always 21000 Gas. In other words, a successful DETER-Z attack can occupy a block space of $16 \cdot 10^6$ Gas (i.e., the block Gas limit) by abusing only 21000 Gas. In terms of Ether cost, as a successful DETER-Z attack needs to make its first transaction's Gas price higher than the prices of all existing transactions in the victim `txpool`, the DETER-Z's Ether cost depends on the `txpool` content. In our experiment, the highest Gas price of existing transactions is 1000 Gwei per Gas, under which DETER-Z attack's Ether cost is reported as in Table 2.

5 ATTACK STRATEGIES AT MINERS

This section presents the attack strategies (§ 5.1) and evaluation (§ 5) on disabling an Ethereum node's mining. There are other Ethereum services beyond mining that depend on the txpool data. We summarize our evaluation results of DETER-ing the other txpool dependent services in § 13.5 while leaving the full details to Appendix 13.

5.1 Proposed Strategies

Given a successfully disabled txpool, the attacker's goal is to disable the victim node's miner, so that the block it finds will be empty.

*Preliminary on miner-*txpool *interaction*: An Ethereum node follows the workflow below to mine the transactions in the local txpool. Upon receiving a block, say b_0, the node evicts from txpool the transactions in b_0 and their dependent ones (e.g., the invalid pending transactions as in DETER-Z). After the miner appends the block b_0 to the tail of its blockchain, it reads the current content of txpool to select the batch of transactions, such as based on Gas prices, to mine for the next block. For instance, a miner reading the txpool at time t_1 may need period dt_4 to find a block, thus the block found at time $t_1 + dt_4$ only contains the transactions in txpool before t_1.

DETER-Z's strategy: To attack a co-residing miner, one can apply DETER-Z in a straightforward way. That is, consider a victim block found at time t_v and its predecessor block propagated at time t_p. A DETER-Z attack can succeed as long as the attacker finishes the attack (Steps z1 and z2) after t_p and before t_v.

DETER-X's strategy: When attacking a miner, a DETER-X attacker needs to ensure the right timing, that is, the submission time of the attacker's request (in sending x1) should be right before when the miner reads txpool, so that the miner reads an empty txpool that is just purged. Denote the time when the miner reads a txpool by t_p. If the attack request occurs after t_p (or long before t_p), chances are the miner will read a non-empty txpool that is not yet purged (or that is refilled by subsequent transactions).

The attack strategy is this: The attacker node watches a local clock. If time T passes on the clock, it sends the attack request and resets the clock. It also resets the clock upon the arrival of a block.

Analysis of attack strategy: It is known that the sequence of block arrivals in a proof-of-work blockchain can be modeled as a Poisson process, under the assumption of constant difficulty and hash rate [49, 59]. Based on the classic probability theory [56], the wait time for the next block is a random variable following exponential distribution. That is, given the average block time T_0, the block arrival rate is $\lambda = \frac{1}{T_0}$. Thus, the wait time for the next block, denoted by x, follows the exponential distribution with density function: $f(x) = \lambda e^{-\lambda x}$. The probability that the next block arrives after time T is the cumulative distribution function of x:

$$Pr[x > t] = e^{-\lambda t} = e^{-\frac{t}{T}}$$

Now consider the initial case that a DETER-X attacker resets his clock.[6] The success rate of the attack is the number of transactions discarded in the miner's next read of txpool divided by all valid transactions received until the next txpool read.

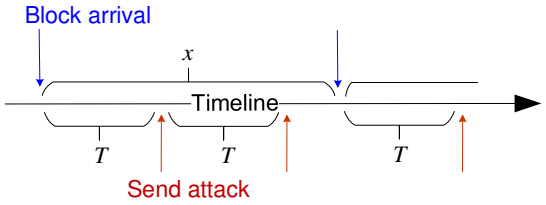


Figure 5: A process with two block arrivals between which the attacker sends two requests. Assuming independent block arrivals, this can be modeled as a Poisson process.

Generally, suppose there are n periods, each lasting T, before the arrival of the next block, as in Figure 5. In that txpool read, $\frac{n \cdot T}{x}$ percentage of the transactions are purged by the n attack requests and $1 - \frac{n \cdot T}{x}$ percentage of transactions are read by the miner. This leads to the success rate blow:

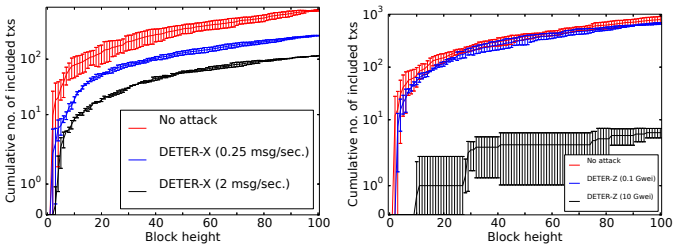
$$S(n) = \frac{n \cdot T}{x} \tag{1}$$

The probability this event occurs (i.e., the next block arrives after the end of n-th period and before the end of $(n+1)$-th period) is:

$$\begin{aligned} Pr(n, T) &= Pr[x > n \cdot T] - Pr[x > (n+1) \cdot T] \\ &= e^{-\frac{nT}{T_0}} - e^{-\frac{(n+1)T}{T_0}} \end{aligned} \tag{2}$$

Hence the expected success rate under attack frequency $\frac{1}{T}$ is:

$$\begin{aligned} E[S] &= \sum_{n=0}^{\infty} Pr(n, T) \cdot \frac{\int_{x=n \cdot T}^{(n+1)T} S(n) dx}{T} \\ &= \sum_{n=0}^{\infty} n \cdot \ln(1 + \frac{1}{n}) Pr(n, T) \end{aligned} \tag{3}$$



(a) Included txs under DETER-X **(b) Included txs under DETER-Z**

Figure 6: DETER a miner in a local network

DETER - attack rate	Success rate		Cost (Ether)
	Measured	Theoretical (Eqn. 3)	
X-.25msg/sec	65.2±4%	61.33%	0
X-2msg/sec	84.9±3%	91.32%	0
Z~ .25tx/sec	99.3±0.1%	100%	0.021/block
Baseline	96.6±0.1%	100%	12.5/block

Table 3: DETER & baseline attacks

[6] There are actually two possible initial cases, the attacker node receiving a newly arrived block or the attacker node having just sent the attack message. As the block arrival events are independent and the exponential distribution is memoryless, we can consolidate the two causes in one initial case.

5.2 Evaluation on a Local Miner Node

The evaluation answers the following research question (RQ):

> **RQ2.** How effective and costly is sending DETER attacks continuously using the strategies presented in § 5.1 in disabling a miner with arriving transactions?

We set up a private network of three nodes: A normal node, an attacker node and a victim miner. The victim is connected to both the normal and attacker nodes. There is no connection between the normal and attacker nodes. In each experiment to be described below, we configure the normal node to generate normal transactions and send them to the victim, at a rate of 2 transactions per second. The process lasts for 500 seconds, and after that, we turn off the mining on the victim node. The metric we use for attack effectiveness is the number of normal transactions included by the victim miner in each block.

In our evaluation of DETER-X, the attacker sends each (x1) message with $n' = 5120$ crafted future transactions to the victim. First, we fix the rate in which regular transactions are generated at 2 transactions per second and vary the attack rate between 0.25 and 2 messages ((x1)) per second. For comparison, we also run the experiment without attacks. We run each experiment with mining 100 blocks and report the cumulative number of transactions included in the blockchain. For instance, at (relative) block height 45, we report the total number of transactions included from the first block to the 45-th block. As mining is a non-deterministic process, we repeat each experiment by three times and report the average cumulative number with its standard deviation in Figure 6a. The result shows that as block height grows, the cumulative number of included transactions increases linearly (note the log scale of Y axis). At the block height of 100, there are 764 transactions included when there are no attacks. By comparison, at the same block height 100, the number of included transactions under DETER-X of 0.25 messages per second (2 messages per second) is 266 transactions (115 transactions), leading to the attack success rate (defined as the percentage of regular transactions discarded in a block) of $1 - \frac{266}{764} = 65.2\%$ ($1 - \frac{115}{764} = 84.9\%$). Statistically speaking, it can be seen over time (at different block heights), there is a consistent disparity between the result with DETER-X attacks and that without. The disparity clearly shows the effectiveness of our DETER-X attack with varying attack rates – The larger the attack rate is, the larger the disparity is.

In our evaluation of DETER-Z, the attacker sends a message of $n' = 5120$ pending transactions described in § 5 , as well as another message of $n' = 5120$ future transactions, to the victim. The attacker sends these two messages every time it receives a new block from the victim. Figure 6b reports the same cumulative metric for DETER-Z, from which one can see DETER-Z is much more effective than DETER-X. Sending approximately 0.25 messages per second, the DETER-Z attack achieves a success rate of 99.3%.

The table in Figure 3 summarizes the measured success rates for the two DETER variants from our previous experiment results. We apply Equation 3 to DETER-X under rates 0.25 and 2 messages per second. The theoretical results roughly fall within the error rates of the measured results. DETER-Z is expected to achieve 100% success rate. In reality, the measured rate is slightly lower than the perfect 100% rate, which we suspect is due to spurious block production (e.g., two blocks are mined at very close time). We also carry out the baseline attack described in the previous subsection (§ 3). The measured success rate is also lower than the expected 100% success rate due to the same suspected reason. At last, in this table, we show the Ether cost of the attacks. DETER-X relies only on future transactions and do not cost the attacker any Ether. In DETER-Z, the attacker needs to pay the fee of one transaction (the first pending transaction) per block, minimally 21000 Gas at 1000 Gwei (i.e., 0.021 Ether). The cost of the baseline attack is 12.5 Ether as previously described. In summary, DETER-X incurs zero monetary cost and achieve reasonable success rates at low attack rate. DETER-Z is very effective with almost 100% success rate and its Ether cost is 1000× cheaper than the baseline attack.

6 MEASURING DEPLOYED NETWORKS' EXPLOITABILITY

This section presents the measurement studies on the exploitability of deployed Ethereum networks, including testnets and the mainnet. We first describe the design rationale in § 6.1 and then the two studies on testnets in § 6.2 and on the mainnet in § 6.3.

6.1 Design Rationale

In the previous sections, our evaluation focuses on the attack effectiveness on Ethereum clients with the default configurations. And the evaluation is set up on a single Ethereum node under our control. This section's goal is to understand a deployed Ethereum network's exploitability under DETER attacks. This is necessarily a different goal (from previous sections), because an attacker who can successfully DETER a controlled node may be hindered when attacking a deployed network: First, in a deployed network, the critical nodes to a network's operation can be hidden from the attacker, leaving her unable to discover the attack targets in the first place. This motivates our first measurement goal, *node discoverability* of an operational network. Second, even the attacker can discover and connect to a critical node, the node may be configured to weaken or mitigate the DETER attacks, for instance, `txpool` can be configured to decline future transactions altogether to be resilient to DETER-X. This motivates our second measurement goal, *node exploitability* of an operational network.

We measure two subjects, testnets and the mainnet. These two types of Ethereum networks serve different purposes in operation and have different levels of impacts to the real world. Concretely, the purpose of testnets is for testing DApps pre production, and they are mainly used by DApp/blockchain developers. The purpose of mainnet is to serve actual DApps in their business, and it is accessed by hundreds of millions of Ethereum DApp users.

Our measurement studies on testnets and the mainnet differ in their measurement methods. For the testnets, we aim to mount DETER attacks directly (with parameters tuned down) and expect to observe some temporary service degradation. By this means, we can produce definitive results regarding exploitability. For the mainnet, our measurement method is designed with ethical concerns as the first-class citizen and aims at lightweight test probes on some necessary (but insufficient) conditions of DETER vulnerabilities.

Particularly, testing the mainnet is necessary, as its exploitability is specific to the client configurations, running discovery protocols, and other deployment-specific settings on the mainnet. Another

network's measurement results cannot be generalized to infer the mainnet's exploitability.

6.2 Measuring Testnets' Exploitability

This subsection presents the measurement study to answer the following research question (RQ).

> **RQ3.** Can a DETER attacker discover the critical service nodes in a testnet (node discoverability)? Can these discovered nodes be effectively attacked (node exploitability)?

6.2.1 Measurement Methods. **Attack Strategy**: With the goal to disable an Ethereum network, the attacker takes the strategy to first discover critical nodes in the network and then direct her DETER payloads at the nodes. Recall that operational blockchains' functionalities are centralized in the hands of "top" services, such as infura.io for transaction relay and Sparkpool for mining blocks. A critical node is defined as the Ethereum node serving the backend of the top services.

Methods to measure node exploitability: In testnets, node exploitability is tested by directly sending DETER payloads (in a short period) and observing the service interruption, such as the block size when the attacked service is a mining pool or the slowdown of transaction propagation when the target is a RPC service. More challenging is measuring node discoverability as described next.

Methods to measure node discoverability: To discover the critical nodes, we propose to leverage the client version "codename" disclosed through the service's frontend RPC interfaces. Specifically, given a known "top" service that exposes the RPC interface, we send `web3_clientVersion` RPC queries and obtain the results, from which we select the unique ones or the ones bound to the specific service (e.g., SrvR1's client codename *omnibus* is unique).

We then launch a sufficient number of "supernodes" to join the mainnet. We configured these supernodes to stripe away their default limit of neighbor numbers. We run these nodes long enough until they are connected the maximal number of nodes in the network (i.e., their neighbor count becomes stable). After that, the supernodes propagate transactions and blocks with their neighbors. During this entire process, the supernodes, as they are statically instrumented, log all the messages that they receive and send, which include the peer-discovery messages, propagated blocks, propagated transactions, among others.

On the collected messages, we find the peer-discovery messages that match the known services' "codenames" (i.e., *omnibus*). A node Nx that sends a peer-discovery message with a matching codename of service S is on the backend of service S.

In addition, when discovering critical nodes behind a mining pools, we analyze the block-arrival messages. Each block-arrival message is a triplet: the block hash, the sender (i.e., the supernode's neighbor that propagates the block), and the timestamp recording when the supernode receives the block-arrival message. Here, we consider both finalized blocks in the blockchain and uncle blocks that are "reorganized" out of the permanent blockchain. For each block, we find its "home" node by choosing the neighbor who sends the block-arrival message at the earliest time. Then, for each given neighbor, we count the number of blocks whose home nodes match this neighbor's nodeID. By this means, we can find top miners in the network as the supernode's neighbors that have produced most blocks in an extended period.

6.2.2 Measurement Results. **Discovering top mining pools**: We apply the above methods to first discover top miners in Ropsten. Specifically, we launch two supernodes joining the testnet and run them for 24 hours until their neighbor sets become stable (i.e., stop growing). In this period, the two supernodes are connected to 840 nodes in Ropsten and receive a total of 6200 distinct blocks. We use the data aggregation described above to plot the distribution of blocks over their home miners as in Figure 7a.

From the result, we choose to the target of DETER attacks (or exploitability tests) the top four miner nodes who jointly produce 88.38% of all blocks. We then respectively mount DETER-X and DETER-Z to attack these selected top miners. Our goal is to understand, by DETER-ing these top miners, how much interrupted the global mining activity of the testnet is.

We similarly set up two other supernodes in connection with 490 nodes in Rinkeby, discover the top miners there and conduct node-exploitability tests. Due to the space limit, we only show the result of attacking Ropsten via DETER-X and leave the results of other settings to Appendix 13.4.

In the above setting, we configure DETER-X by sending $n' = 5120$ future transactions with a Gas price of 1000 Gwei at the rate of 1 message per second. The DETER-X attacks to the selected top miners are mounted in parallel. To control the service interruption, we restrict the duration of attack in Ropsten under 60 seconds. After the attack, we took a screenshot of the monitored block history from etherscan.io [28].

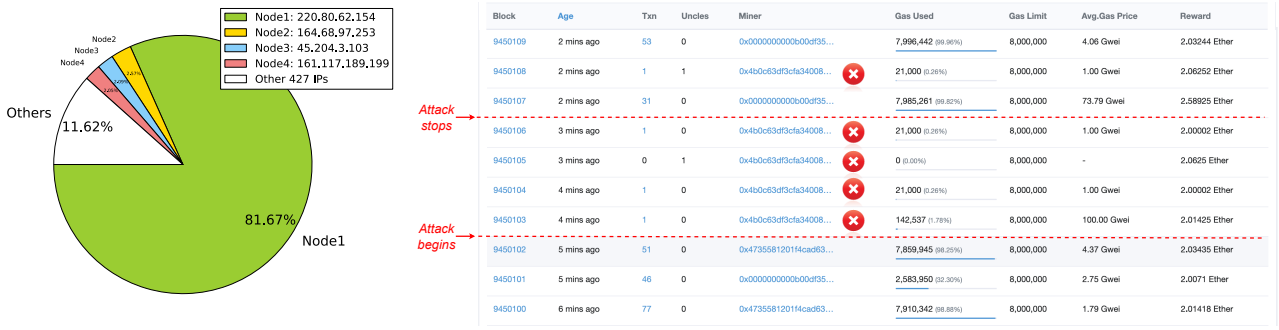
In the screenshot in Figure 7b, our attack starts at block #9450103 and stops at block #9450106. Before the attack starts (blocks #9450100-9450102), each block includes at least 46 transactions and the used Gas in each block varies between 32.3% and 98.9% (normalized over the Ethereum block Gas limit). During the attack (blocks #9450103-#9450106), each block includes at most 1 transaction and the used Gas per block is below 1.8%. Note that these four blocks are mined by miners under DETER-X attacks. After the attack stops (blocks #9450107-#9450109), for blocks #9450107 and #9450109, each block includes at least 31 transactions and has a used Gas per block above 99%. These two blocks are found by the miners not under our DETER-X attacks. Block #9450108 includes only one transaction and is mined by the miner under attack. Note that even though the attack stops when Block #9450108 is produced, it is likely that its home miner $0x4b0c\ldots$ reads `txpool` before the attack stops, and it reads a purged `txpool`.

Overall, by comparing the numbers before/during/after the attack, we show the DETER-X attack of 0 Ether cost can reduce the block size in Ropsten by $31-77\times$ in terms of the number of included normal transactions, and by $18-55\times$ in used Gas per block.

6.3 Measuring Mainnet's Exploitability

This subsection presents the measurement study to answer the following research question (RQ).

> **RQ4.** Can a DETER attacker discover the critical service nodes in the mainnet (node discoverability)? Can these discovered nodes be effectively attacked (node exploitability)?



(a) Top miners in Ropsten

(b) Screenshot from etherscan.io: red cross indicates the miners under DETER-X attack.

Figure 7: Evaluation of DETER-X attack on the top-4 miners in the Ropsten testnet.

6.3.1 Measurement Method. **Design rational**: Our goal is to measure the DETER exploitability of an identified mainnet node. A naive approach is to directly run the original test t_1/t_2 (recall § 4) against the mainnet node. Unfortunately, this does not work for a mainnet node which we don't have control. Specifically, measuring DETER vulnerability entails setting up the `txpool` with certain initial transactions and observing an eviction by incoming future/invalid transactions on the target node. Directly carrying out these actions requires privilege (e.g., turning on RPC interface as in t_1/t_2), which we don't have if the target is a mainnet node operated by others. Moreover, the admission of a future transaction (as in t_1) and the eviction by a latent invalid transaction may only change the internal state and is not externally observable.[7] Besides, even if an eviction can be detected on a mainnet node, attribution to the right cause can be challenging, as a mainnet node, unlike a local node in a controlled environment (in § 4.4), needs to also process normal transactions propagated in the "background." An eviction can be attributed to a background transaction or a test transaction.

To address the measurement challenges, our idea is two-fold: 1) Instead of observing the opaque future transaction directly, we send a pending transaction to be evicted by the future transaction and observe its behavior instead. As the pending transaction is propagated, it is observable across nodes. 2) Instead of relying on RPC service that we cannot configure on the mainnet, we exploit the *transaction replacement* capability in the standard Ethereum protocol: An incoming transaction tx_2 may replace an existing transaction tx_1 of the same sender and nonce, if tx_2's Gas price is sufficiently higher than tx_1's (e.g., higher than 110% of tx's Gas price in Geth). The insight here is that *when sending a tx_2 at price lower than* 110% *of tx_1's Gas price, say* 105%*, observing the propagation of tx_2 on other nodes implies that* `txpool` *does not initially contain tx_1.*

To ensure that 105% is low enough, we profiled all existing Ethereum clients. Denote by R the minimal Gas price bump necessary to replace an existing transaction. Table 4 shows the profiling results, in which all clients' R are above 105%, which is also consistent with existing works [67].

Our design also addresses the new challenges presented by ethical measurement on the mainnet. Instead of purging the entire `txpool` and victimizing all residing transactions there, our mainnet test is designed to affect only the transactions with too low Gas price to be included in the blockchain (within the default three-hour drop deadline).

Table 4: Gas price bumps in different Ethereum clients.

	Geth	Parity	Nethermind	Besu
R	110%	112.5%	110%	110%

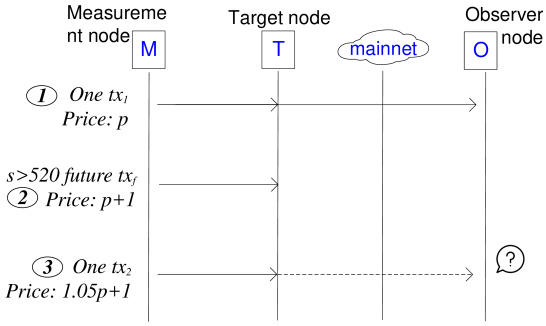


Figure 8: Running test t_{1m} against a mainnet target node.

Mainnet test t_{1m}: Suppose the target is to measure the DETER-X exploitability of a mainnet node T. Initially, we set up a measurement node M and an observer node O, separately connecting to T. We double-check if node M's `txpool` is full, and only proceed upon a full`txpool`. In ①, Node M sends a pending transaction tx_1 of medium Gas price p such that p is higher than at least s_0 (e.g., $s_0 = 520$) existing transactions in Node M's `txpool`. Note the `txpool`'s capability is $n = 5120$ transactions. Node O checks if tx_1 is propagated by node T. It proceeds only if this is true. In ②, Node M sends to Node T s future transactions, each at price $p + 1$. The number of future transactions $s > s_0 = 520$. In our measurement, Node M runs a statically instrumented Geth client so that it can propagate future transactions to T. In ③, immediately after the previous step, Node M sends Node T one replacing transaction tx_2, whose price is $1.05 \cdot p + 1$. At last, Node O observes if it receives tx_2 from node T. If so, the test is a success , denoted by ✓, which indicates that Node T does evict tx_1 by one of those future transactions. We denote this test by $t_{1m}(s, p) =$✓|✗. An illustration of the measurement process is in Figure 8.

Measurement effectiveness: Consider the situation right after Step ③. If tx_1 is evicted by one of the future transactions, tx_2 will

[7] By contrast, admission of a pending transaction to the `txpool` is externally observable, as that results in the admitted pending transaction propagated to other nodes.

arrive as a new transaction which will be admitted to the txpool and propagated to T's neighbors, including Node O. If tx_1 is not evicted, tx_2 will arrive as a transaction attempting to replace tx_1 which will fail because of its Gas price $1.05 \cdot p + 1 < 1.1 \cdot p$. Thus, in this case, Node O cannot observe the propagation of tx_2.

In practice, when there are normal transactions propagated to the target mainnet node, the normal transactions are assumed to arrive at a volume much lower than the burst of s future transactions in the test. Thus, a normal transaction may evict one of the s_0 existing transactions at lower Gas prices but not tx_1.

Mainnet test t_{2m}: We propose a mainnet version of test t_2, named t_{2m}. Similar to t_{1m}, the measurement node M first propagates to the target mainnet node x_2 transactions $tx_{1,0}, tx_{1,1}, \ldots tx_{1,x_2}$ at price p higher than s other transactions in the txpool, then sends a message of s latent invalid transactions at price $p + 1$, and at last sends x_2 replacing transactions $tx_{2,0}, tx_{2,1}, \ldots tx_{2,x_2}$ at price $1.05 \cdot p + 1$. The observer node O observes the replacing transactions that are propagated to it. If all (or less than) x_2 transactions are propagated, the test is a full (or partial) success, denoted by ✓ (or ✓). Otherwise, if none of the x_2 transactions are propagated, the test fails, denoted by ✗.

Ethical designs: We take extensive measures to address the ethical concerns. First, the measurement methods presented above limit the prices of crafted transactions to be lower than p, thus leaving the existing pending transactions on the tested node (i.e., node T) unaffected during and after the test. Recall price p is set s.t. 10% of existing transactions on Node T have lower prices than p.

Second, to minimize the impacts on the 10% pending transactions on the tested node T, we sent the these transactions after each test to "refill" the txpool. To do so, in our studies, we increase the length of the measurement node's txpool to ensure enough transactions are buffered there.

Third, we check 1) the blocks generated during and right after each test are full and reach the block Gas limit, and 2) the lowest Gas price of the transactions in those blocks is higher than p. These two conditions jointly ensures that the presence of the test, which affects at most 10% of the transactions in the txpool, does not affect the transactions included in the blocks, thus leaving no long-term effect on the blockchain.

6.3.2 Measurement Results. **Node discoverability in RPC services**: To discover the critical nodes in the mainnet, we use the same method described in § 6.2.1, only with different configurations as described next. We send web3_clientVersion RPC queries to eight well-known RPC services (including infura [9], blockdaemon [4], etherscan [28], quiknode [14], et al), and find that SrvR1 and SrvR2 nodes[8] bear unique codenames.

We then launch eight "supernodes" to join the mainnet and run them in a 7-day period until the neighbors of each supernode stop growing and are stable. Here, launching eight supernodes increases the node coverage in the mainnet. Using the measurement methods described in § 6.2.1, we discover 48 nodes serving the backend of SrvR1 and 1 node of SrvR2, as presented in Table 5.

Node discoverability in mining pools: To discover the top miners on the mainnet, we reuse the same methods with that of the testnets as described in § 6.2.1. We reuse the eight mainnet supernodes in the previous experiment. Specifically, we first check if a mining pool provides a RPC service by visiting its website. We did so for all top mining pools listed on the ranking website [11]. If a RPC service is provided, we use the method described previously to discover the mining pool's codename. We found mining pools SrvM1's [17] nodes run Geth clients with codename *turbo*, and SrvM2' [22] on clients of two codenames, *ethereumsolo* and *ethereumpplns*. Other mining pools' codenames are strongly suggestive w.r.t. their names. Then, we use the found codename to match the peer-discovery messages collected through the eight supernodes and discover the neighbor nodes that serve the backend of known mining pools. Additionally, the supernodes monitor the block-propagation messages and their timings to verify the discovered top mining-pool nodes.

Table 5: Critical mainnet nodes: Discoverability and exploitability (We anonymize the two services' labels by XX).

Service name	# of nodes	t_{1m}/X	t_{2m}/Z	Client-codename
Mining pools				
SrvM1	59	✓	✓	Geth-*turbo*
SrvM2	8	✓	✓	Geth-*ethereumsolo*, Geth-*ethereumpplns*
SrvM3	6	✓	✓	Geth-XX
SrvM4	2	✓	✓	Geth-XX
RPC services				
SrvR1	48	✓	✓	Geth-*omnibus*
SrvR2	1	✓	✓	Geth-*ethshared*

After searching the codenames in the peer-discovery messages collected, we found 59 nodes in SrvM1, 8 nodes in SrvM2 and 6 nodes of SrvM3, as listed in Table 5. Note in our experiments, if waiting for long enough, the measurement node can always be connected to the nodes of the target mining pool. Particularly, we have not found mining pools that only connect to a prefixed set of nodes.

Node exploitability: For each identified critical node (from § 6.3.2), we run the above test t_{1m} and t_{2m}, respectively for measuring DETER-X and DETER-Z exploitability. As all identified nodes run Geth clients, we use the Geth-default setting in our tests, such as $n = 5120$. For each service, we pick two random nodes to the two tests, and each test is run for three times to ensure the same result is produced. We report the result in the two columns named "t_{1m}/X" and "t_{2m}/Z" in Table 5. All tests are success, and all tested nodes are vulnerable under both DETER-X and DETER-Z attacks.

7 ATTACK MITIGATION

Due to the impossibility result discussed in § 1, we propose heuristics for DETER mitigation and miners' profitability preservation. We propose two mitigation schemes M_0 and M_1 that respectively add restriction to transaction admission and eviction policies. Due to the space limit, we put M_0 in Appendix 14.1. What follows is mitigation scheme M_1.

Mitigation scheme M_1: We add three transaction-eviction rules over the underlying txpool: M_{1a}) It declines an incoming future transaction if it evicts a valid pending transaction in the txpool. M_{1b}) It declines an incoming invalid transaction if it evicts a valid pending transaction in the txpool. M_{1c}) It declines an incoming transaction if it evicts another pending transaction and leaves a future transaction in the txpool. For instance, consider two pending

[8] We anonymize the names of these services that we conduct tests on.

transaction tx_1 of nonce $n+1$ and tx_2 of nonce $n+2$. If tx_1 is evicted, it makes tx_2 a future transaction. Such an eviction is prohibited by M_{1c}). M_{1d}) Other than the above, it optionally enforces the following eviction priority, that is, $VH > VL > [FI]H > [FI]L$ (valid transaction preceding future or invalid transactions, even their Gas prices disagree). Note that M_1 does not restrict transaction admission on a `txpool` if it does not trigger eviction.

Security analysis: With M_{1a}), a DETER-X payload will be declined. With M_{1b}), a DETER-Z payload will be declined. M_{1a}) and M_{1b}) work for generic initial state of a `txpool`. M_{1c}) defends against DETER attacks in the special case where incoming transactions trigger transforming existing transactions into future transactions. In addition to security analysis, we evaluate M_1's security by implementing it in our `txpool` simulator (as will be described) and measure the success rate of DETER attacks on it.

Schemes	Miners' revenue (Ether)	DETER security	
		t_1/X	t_2/Z
Geth (default)	16.5388	✓/✓ (Table 2)	
M_0 (in Appendix 14.1)	15.9506(−3.56%)	✗	✗
M_1	16.5423(+0.002%)	✗	✗

Table 6: Evaluation of miners' revenue and security across mitigation schemes and Ethereum clients: All clients are configured with the same `txpool` length of 5120 transactions.

Table 7: Transaction statistics during the evaluations on mitigation schemes. "#" is number, "txs" refer to transactions, and conditions F/I mean future/latent invalid transactions being declined. Conditions $FH \rightarrow VL/IH \rightarrow VL$ mean the declined transactions are future/latent invalid transactions with high Gas prices that, if not declined, would have evicted pending transactions. Statistics labeled by – are not collected in our experiments due to the lack of usefulness.

Schemes	# of declined txs (F)	# of declined txs (I)	# of declined txs ($FH \rightarrow VL$)	# of declined txs ($IH \rightarrow VL$)
Geth	1395	1589	0	0
M_0	9588	1899	–	–
M_1	288	353	287	322

Evaluation of mitigation schemes: We evaluate mitigation schemes in terms of miners' revenue and the success rates of DETER attacks. The two mitigation schemes are implemented on Geth and are evaluated under real transaction traces. We use Geth's default transaction-admission policy as the baseline for comparison. Due to the space limit, we leave other experiments, such as miners' revenue with synthetic transactions and on other Ethereum clients, to the Appendix 14.

For security evaluation, we run tests t_1 and t_2 with varying parameters (x_1 and x_2) and observing the test results on the two mitigation schemes. As shown in Table 6, all tests under all parameters fail with ✗ (i.e., zero success rates), suggesting the effectiveness of mitigating DETER attacks.

For miners' revenue, we launched a mainnet node and collect the Ethereum transactions received there. We then replay the trace against a Geth node on which we build a middleware to simulate an additional `txpool` and implement the proposed mitigation schemes in the middleware. The `txpool` simulator's length is set at 5120, which is the same with Geth's default `txpool` length. On different clients/experiments, we replay the same sequence of interleaved transactions and mining actions. Each experiment of the same setting is run three times and we verify that the replayed runs produce the same results deterministically. All produced blocks in the experiments are full and with the same block Gas limits. We report the total Ether of all transactions included in these blocks. The details of the experiment setup is in Appendix 14.2.1. During the experiments, we also collect additional workload statistics to help explain results. We collect the number of declined future/latent invalid transactions (i..e, F/I) and the number of declined future/latent invalid transactions that, had not declined, would have evicted a pending transaction at a lower Gas price (i.e., $FH \rightarrow VL/IH \rightarrow VL$). These statistics are obtained on the Geth node under our control.

The results of miners' revenue are presented in Table 6. M_0's revenue is 3.56% lower than the baseline of Geth's default policy, and M_1's revenue is almost the same with Geth's (0.0002% higher). The statistics in Table 7 help understand the revenue results. From the table, M_0 declines much more transactions (both F and I) than the vanilla Geth, thus failing to collect the potential revenues from those transactions and resulting in lower revenue than Geth's. M_1 declines fewer transactions than Geth and may benefit its revenues from those transactions' fees. Note that among those declined transactions in M_1, majority are the ones exploitable by DETER attacks, that is, $[FI]H \rightarrow VL$, and are necessary to be declined.

8 RESPONSIBLE DISCLOSURE

We have disclosed the DETER vulnerabilities to the Ethereum developer community of Geth/Parity/Besu/Nethermind[9] (through their bug bounty programs), as well as tested service providers (including the RPC services and mining pools). The bugs have been confirmed by all clients' bug-bounty programs with attacks reproduced. Particularly, the DETER bugs are assessed to be of "high impact" by Ethereum Foundation (Geth) and "median impact" by OpenEthereum (Parity).

9 CONCLUSION

This work presents the DETER attacks that deny a remote Ethereum node's service by exploiting flawed transaction handling in `txpool`. DETER attacks are of low Ether cost. DETER attacks can be extended by discovering nodes in critical services and result in a global impact on an Ethereum network. We evaluate and verify the effectiveness and low cost of DETER attacks on local nodes running different Ethereum clients and testnets. We also propose non-trivial methods to detect DETER vulnerability on blackbox mainnet nodes and confirm the mainnet nodes' discoverability and exploitability.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers in ACM CCS'21 and USENIX Security'21. The authors appreciate the discussion with Xianghong Liu in the early stage of this work. The authors are partially supported by the National Science Foundation under Grants CNS1815814 and DGE2104532.

[9]We also send bug reports to Aleth, but with no response.

REFERENCES

[1] Litecoin cash (lcc) was 51% attacked. https://bit.ly/36EUs7p, .

[2] Aleth – ethereum c++ client, tools and libraries. https://github.com/ethereum/aleth, .

[3] Hyperledger besu. https://www.hyperledger.org/use/besu, .

[4] Blockdaemon: Institutional grade blockchain infrastructure. https://blockdaemon.com/, .

[5] Json-rpc in ethereum wiki (eth_call). https://github.com/ethereum/wiki/wiki/json-rpc#eth$_c$all, .

[6] Irreversible transactions: Finney attack. https://en.bitcoin.it/wiki/Irreversible$_T$ransactions#Finney$_a$ttack, .

[7] Gas now: Eth gasprice forecast system based on sparkpool pending transaction mempool. https://www.gasnow.org/, .

[8] Geth: the go client for ethereum. https://www.ethereum.org/cli#geth, .

[9] Ethereum & ipfs apis. develop now on web 3.0. https://infura.io/, .

[10] Infura mainnet outage post-mortem 2020-11-11. https://blog.infura.io/infura-mainnet-outage-post-mortem-2020-11-11/, .

[11] Ethereum directory for mining on etherscan. https://etherscan.io/directory/Mining, .

[12] Nethermind ethereum client. https://nethermind.io/client, .

[13] Parity ethereum is now openethereum: Fast and feature-rich multi-network ethereum client. https://www.parity.io/ethereum/, .

[14] Blockchain infrastructure cloud. https://www.quiknode.io/, .

[15] The rinkeby testnet of ethereum. https://rinkeby.etherscan.io, .

[16] The ropsten testnet of ethereum. https://ropsten.etherscan.io, .

[17] Sparkpool: Crypto mining and staking pool. https://www.sparkpool.com/en/, .

[18] Go ethereum's txpool namespace. https://geth.ethereum.org/docs/rpc/ns-txpool, .

[19] Memoria 700 million stuck in 115,000 unconfirmed bitcoin transactions. https://www.ccn.com/700-million-stuck-115000-unconfirmed-bitcoin-transactions/, .

[20] Geth nodes under attack again (reddit). https://www.reddit.com/r/ethereum/comments/55s085/geth$_n$odes$_u$nder$_a$ttack$_a$gain$_w$e$_a$re$_a$ctively/, .

[21] The Trinity client for ethereum network. no longer maintained or developed. https://github.com/ethereum/trinity, Retrieved Aug. 20, 2021.

[22] Altcoin mining pool for gpu and asic: 2miners. https://2miners.com/, Retrieved May, 5, 2021.

[23] Bitcoin gold hack shows 51% attack is real. https://www.investopedia.com/news/bitcoin-gold-hack-shows-51-attack-real/, Retrieved May, 5, 2021.

[24] Report: Bitcoin (btc) mempool shows backlogged transactions, increased fees if so? https://goo.gl/LsU6Hq, Retrieved May, 5, 2021.

[25] Dapp survey results 2019. https://medium.com/fluence-network/dapp-survey-results-2019-a04373db6452, Retrieved May, 5, 2021.

[26] Known attacks - ethereum smart contract best practices. https://consensys.github.io/smart-contract-best-practices/known$_a$ttacks/#dos-with-block-gas-limit, Retrieved May, 5, 2021.

[27] Ethereum mainnet statistics. https://www.ethernodes.org/, Retrieved May, 5, 2021.

[28] Etherscan: Ethereum (eth) blockchain explorer. https://etherscan.io/, Retrieved May, 5, 2021.

[29] Eth gas station: Recommended gas prices in gwei. https://ethgasstation.info/, Retrieved May, 5, 2021.

[30] ethgasstation:an adaptive gas price oracle for the ethereum blockchain. https://github.com/ethgasstation/ethgasstation-backend, Retrieved May, 5, 2021.

[31] Eth gas station: Txpool report. https://ethgasstation.info/txPoolReport.php, Retrieved May, 5, 2021.

[32] Optimism's tweet of apology for purposefully triggering the ethereum bug. https://twitter.com/jinglanW/status/1326651349912719360, Retrieved May, 5, 2021.

[33] Taichi network. https://taichi.network/, Retrieved May, 5, 2021.

[34] Taichi network's block propagator. https://github.com/Taichi-Network/docs/blob/master/deploy.md, Retrieved May, 5, 2021.

[35] Taichi network's rpc service (asia pacific). https://api.taichi.network:10001, Retrieved May, 5, 2021.

[36] Transaction spam attack: Next steps. https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/, Retrieved May, 5, 2021.

[37] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In IEEE Symposium on SP 2017, pages 375–392, 2017. doi: 10.1109/SP.2017.29. URL https://doi.org/10.1109/SP.2017.29.

[38] Khaled Baqer, Danny Yuxing Huang, Damon McCoy, and Nicholas Weaver. Stressing out: Bitcoin "stress testing". In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers, volume 9604 of Lecture Notes in Computer Science, pages 3–18. Springer, 2016. doi: 10.1007/978-3-662-53357-4\$_1$. URL https://doi.org/10.1007/978-3-662-53357-4$_1$.

[39] Joseph Bonneau. Why buy when you can rent? - bribery attacks on bitcoin-style consensus. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers, volume 9604 of Lecture Notes in Computer Science, pages 19–26. Springer, 2016. doi: 10.1007/978-3-662-53357-4\$_2$. URL https://doi.org/10.1007/978-3-662-53357-4$_2$.

[40] Vitalik Buterin. Eip150: Gas cost changes for io-heavy operations. URL https://github.com/ethereum/EIPs/blob/master/EIPS/eip-150.md.

[41] Zhen Cheng, Xinrui Hou, Runhuai Li, Yajin Zhou, Xiapu Luo, Jinku Li, and Kui Ren. Towards a first step to understand the cryptocurrency stealing attack on ethereum. In RAID 2019, pages 47–60, 2019. URL https://www.usenix.org/conference/raid2019/presentation/cheng.

[42] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. CoRR, abs/1904.05234, 2019. URL http://arxiv.org/abs/1904.05234.

[43] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In 2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020, pages 910–927. IEEE, 2020. doi: 10.1109/SP40000.2020.00040. URL https://doi.org/10.1109/SP40000.2020.00040.

[44] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In FC 2014, Christ Church, Barbados, pages 436–454, 2014. doi: 10.1007/978-3-662-45472-5$_2$8. URL https://doi.org/10.1007/978-3-662-45472-5$_2$8.

[45] Amir Feder, Neil Gandal, J. T. Hamrick, and Tyler Moore. The impact of ddos and other security shocks on bitcoin currency exchanges: evidence from mt. gox. J. Cybersecur., 3(2):137–144, 2017. doi: 10.1093/cybsec/tyx012. URL https://doi.org/10.1093/cybsec/tyx012.

[46] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. Decentralization in bitcoin and ethereum networks. CoRR, abs/1801.03998, 2018. URL http://arxiv.org/abs/1801.03998.

[47] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In Jaeyeon Jung and Thorsten Holz, editors, USENIX Security 2015, Washington, D.C., USA, pages 129–144. USENIX Association, 2015. URL https://www.usenix.org/conference/usenixsecurity15.

[48] Benjamin Johnson, Aron Laszka, Jens Grossklags, Marie Vasek, and Tyler Moore. Game-theoretic analysis of ddos attacks against bitcoin mining pools. In Financial Cryptography and Data Security - FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers, pages 72–86, 2014. doi: 10.1007/978-3-662-44774-1\$_6$. URL https://doi.org/10.1007/978-3-662-44774-1$_6$.

[49] Harry A. Kalodner, Malte Möser, Kevin Lee, Steven Goldfeder, Martin Plattner, Alishah Chator, and Arvind Narayanan. Blocksci: Design and applications of a blockchain analysis platform. In Srdjan Capkun and Franziska Roesner, editors, 29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020, pages 2721–2738. USENIX Association, 2020. URL https://www.usenix.org/conference/usenixsecurity20/presentation/kalodner.

[50] Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. Timelocked bribes. IACR Cryptol. ePrint Arch., 2020:774, 2020. URL https://eprint.iacr.org/2020/774.

[51] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. Measuring ethereum network peers. In Proceedings of IMC 2018, pages 91–104, 2018. URL https://dl.acm.org/citation.cfm?id=3278542.

[52] Kai Li, Jiaqi Chen, Xianghong Liu, Yuzhe Richard Tang, XiaoFeng Wang, and Xiapu Luo. As strong as its weakest link: How to break blockchain dapps at RPC service. In 28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021. The Internet Society, 2021. URL https://www.ndss-symposium.org/ndss-paper/as-strong-as-its-weakest-link-how-to-break-blockchain-dapps-at-rpc-service/.

[53] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. IACR Cryptology ePrint Archive, 2018:236, 2018. URL http://eprint.iacr.org/2018/236.

[54] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, Financial Cryptography and Data Security - FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers, volume 10958 of Lecture Notes in Computer Science, pages 3–18. Springer, 2018. doi: 10.1007/978-3-662-58820-8\$_1$. URL https://doi.org/10.1007/978-3-662-58820-8$_1$.

[55] Michael Mirkin, Yan Ji, Jonathan Pang, Ariah Klages-Mundt, Ittay Eyal, and Ari Juels. Bdos: Blockchain denial of service, 2019.

[56] Michael Mitzenmacher and Eli Upfal. Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, 2005. ISBN 978-0-521-83540-4. doi: 10.1017/CBO9780511813603. URL https://doi.org/10.1017/CBO9780511813603.

[57] Daniel Pérez and Benjamin Livshits. Broken metre: Attacking resource metering in EVM. In 27th Annual Network and Distributed System Security Symposium,

NDSS 2020, San Diego, California, USA, February 23-26, 2020. The Internet Society, 2020. URL https://www.ndss-symposium.org/ndss-paper/broken-metre-attacking-resource-metering-in-evm/.
[58] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? CoRR, abs/2101.05511, 2021. URL https://arxiv.org/abs/2101.05511.
[59] Meni Rosenfeld. Analysis of hashrate-based double spending. CoRR, abs/1402.2009, 2014. URL http://arxiv.org/abs/1402.2009.
[60] Muhammad Saad, Laurent Njilla, Charles A. Kamhoua, Joongheon Kim, DaeHun Nyang, and Aziz Mohaisen. Mempool optimization for defending against ddos attacks in pow-based blockchain systems. In IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2019, Seoul, Korea (South), May 14-17, 2019, pages 285–292. IEEE, 2019. doi: 10.1109/BLOC.2019.8751476. URL https://doi.org/10.1109/BLOC.2019.8751476.
[61] Christof Ferreira Torres, Ramiro Camino, and Radu State. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. CoRR, abs/2102.03347, 2021. URL https://arxiv.org/abs/2102.03347.
[62] Muoi Tran, Inho Choi, Gi Jun Moon, Anh V. Vu, and Min Suk Kang. A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network. In To appear in Proceedings of IEEE Symposium on Security and Privacy (IEEE S&P), 2020.
[63] Itay Tsabary, Matan Yechieli, and Ittay Eyal. MAD-HTLC: because HTLC is crazy-cheap to attack. CoRR, abs/2006.12031, 2020. URL https://arxiv.org/abs/2006.12031.
[64] Marie Vasek, Micah Thornton, and Tyler Moore. Empirical analysis of denial-of-service attacks in the bitcoin ecosystem. In Financial Cryptography and Data Security - FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers, pages 57–71, 2014. doi: 10.1007/978-3-662-44774-1_5. URL https://doi.org/10.1007/978-3-662-44774-1_5.
[65] Fredrik Winzer, Benjamin Herd, and Sebastian Faust. Temporary censorship attacks in the presence of rational miners. In 2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019, pages 357–366. IEEE, 2019. doi: 10.1109/EuroSPW.2019.00046. URL https://doi.org/10.1109/EuroSPW.2019.00046.
[66] Wuqi Zhang, Lili Wei, Shuqing Li, Yepang Liu, and Shing-Chi Cheung. Ðarcher: Detecting on-chain-off-chain synchronization bugs in decentralized applications. CoRR, abs/2106.09440, 2021. URL https://arxiv.org/abs/2106.09440.
[67] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V. Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. CoRR, abs/2009.14021, 2020. URL https://arxiv.org/abs/2009.14021.

Appendices

10 RELATED WORKS

Blockchain DoS security: In the existing literature, there is a body of research work on examining the blockchains' security under DoS attacks. These DoS attacks exploit the vulnerabilities at a blockchain's different system layers, such as exploiting underpriced smart-contract execution [26, 40], misusing mining incentives [6, 55], abusing transaction processing [6, 55], as well as partitioning the underlying P2P networks [37, 47, 53, 62]. Specifically, 1) On the mining layer, classic mining algorithms are designed under the assumption that majority of miner nodes are honest, which may not hold in the case of small-scale blockchains. In practice, 51% attacks have been successful on smaller blockchains [1, 23], resulting in damages ranging from hard forks to the demise of entire blockchain. A block-withholding attack works by a miner withholding the block she finds and revealing it later, in a strategic way that wastes the efforts of other miners or mining pools [6]. BDoS [55] exploits selfish mining [44] to incentivize miners to stop mining altogether. In general, the existing mining-based DoS attacks assume institutional attackers who control a significant portion of computing power in a blockchain (e.g., the early-day 51% and the 21% in the very recent BDoS [55]), which may not be practical in targeting a large-scale blockchain network. In practice, there are no successful instances of mining-based DoSes on large blockchains [55]. 2) On the P2P network layer, an eclipse attack [47, 53] aims to isolate a DoS-victim peer from the network. A routing attack [37, 62] assumes corrupted institutional attackers controlling ISP and employs BGP hijacking to intercept network traffic towards partitioning it. 3) On the smart-contract and DApp layer, one can exploit underpriced EVM instructions (e.g., `EXCODESIZE` [36], `SUICIDE` [20] or instruction runtime variation as in BrokenMetre [57]) as well as zero-priced Ethereum operations (e.g., `eth_call` [5] as in the DoERS attack [52]) to cause a large volume of computations on victim EVM instances at low costs to the attacker. In addition, contract execution can be failed by overflowing the call stack [40] or overflowing the block gas limit [26]. Failing a contract call is of interest to, for instance, a malicious auction leader who wants to fail a refund call to the previous leader.

Of particular relevance is the small body of research on 4) the attacks to deny pre-blockchain transaction handling, by sending Bitcoin spams. Bitcoin "stress testing" [38] is a measurement study on the impacts of the 2015 Bitcoin spam campaign. In this campaign, a flood of "spam" transactions (each at a low cryptocurrency value) is broadcast to the Bitcoin network and saturate the limited blockchain transaction throughput [19]. The impact of the campaign includes the transaction-inclusion backlog (victimizing other concurrent transactions and causing delays), enlarged memory pool (which psychologically causes subsequent transaction senders to pay a higher fee than they should [24, 60]), the increased UTXO set (victimizing the miners who need to maintain the full set of UTXO for transaction validation), As the spam transactions are at a low fee and have a low priority for blockchain inclusion, they incur low monetary costs to the attacker. Unlike the Bitcoin spam attacks, DETER is the first work to deny Ethereum network services by exploiting previously unknown design flaws in Ethereum's transaction handling. It also presents a measurement study on the DETER's impacts on real Ethereum clients/networks.

Blockchain RPC attacks: There are research works exploiting or aiming at modern blockchains' RPC services. The work [41] measures the prevalence of cryptocurrency stealing attacks exploiting Ethereum's RPC to unlock accounts, via deploying honeypots. DoERS [52] exploits the free-of-charge RPC interface (`eth_call`) to cripple the Ethereum nodes run by a RPC service. DETER can break a RPC service but exploit the Geth design flaws which are different from the vulnerabilities in the existing RPC attacks.

In-band bribery attacks [39, 50, 54, 63, 65] refers to the attack in which an average attacker account sets up a smart contract with deposit to reward any miner who delays the inclusion of the transactions that modify a victim account' state after a prefixed timeout. In this attack, the miner is assumed to be rational and can choose which arriving transactions to include in order to maximize her revenue, eventually.

In-band bribery attack can be low-cost and be mounted by the average user, instead of an institutional user [50, 63, 65]. This is similar to the DETER attacks. Other than that, the two attacks are different: First, their attack methods are different. While in-band bribery attack allows miners to see the victim transactions (but misuse rational miners' incentive to exclude them temporally), DETER prevents the miners from seeing victim transactions in the first place. Thus, DETER does not require miners to be rational. Second, the consequences of these two attacks are different. While in-band bribery temporally censors the inclusion of selected transactions

sent from few victim accounts (the number of victim accounts can not be big to keep the bribe or attacker's cost low), DETER attacks are to evict *all* transactions submitted during the attack period and cause them to be permanently excluded from the blockchain.

11 DETER-Z CUSTOMIZATION ON ALETH

We design the following test t_4 to check if a txpool admits active (non-latent) invalid transactions, and we test all five Ethereum clients.

t_4 Test $t_4(x_2 \cdot VL + (n-x_2) \cdot IM, IH) \rightarrow$ ✓|✗. The initial state S contains x_2 valid pending transactions at low Gas price (VL) and $n - x_2$ invalid pending transactions at medium Gas price (IM). Each of the IM'es is an overdraft transaction (i.e., spending exceeds the balance) sent from a distinct account. The incoming transaction tx is an overdraft transaction at a high Gas price IH. The success of the test, denoted by ✓ indicates that the overdraft pending transaction ($\{tx\} = \{IH\}$) can evict a valid pending transaction ($\{tx'\} = \{VL\}$). Otherwise, the test result is failure denoted by ✗.

Table 8: Profiling different Ethereum clients under tests and DETER attacks. For the two DETER attacks, success rates are reported. DETER-Z's Ether cost is also reported in parenthesis. Note that the baseline attack incurs a cost of 12.5 Ether per block. The second column refers to the percentage of mainnet nodes running a specific client [27].

Ethereum clients	Percentage	t_1	t_4	t_2	X	Z (Ether)
Geth	83.24%	✓ ($x_1 < 1024$)	✗	✓ ($x_3 < 4096$)	100%	100% (0.021)
		✗ ($x_1 \geq 1024$)		✗ ($x_3 \geq 4096$)		
Parity	14.57%	✓ ($x_1 \geq 2000$)	✗	✓ ($x_3 \leq 81$)	75.6%	100% (2.1)
		✗ ($x_1 \leq 2000$)		✗ ($x_3 > 81$)		
Nethermind	1.53%	✓	✗	✓ ($x_3 \leq 17$)	100%	100% (1.28)
				✗ ($x_3 > 17$)		
Besu	0.52%	✓	✗	✓	100%	100% (0.021)
Aleth	0%	✓	✓	✗ ($x_3 > 1$)	100%	100% (0)

Test t_4 succeed on Aleth but fail on other clients. We believe Aleth enables successful t_4 tests because admitting invalid high-priced transactions is profitable, under certain workloads, to miners.

DETER-Z variant on Aleth: As can be seen, test t_2 does not succeed on Aleth, as it enforces a very restrictive limit: only one transaction per sender is allowed on a full `txpool`. In this case, the attacker can exploit the test result t_4, as described below.

z1': The DETER-Z attacker sends the target Aleth node n' overdraft transactions from n' distinct accounts and at a much higher Gas price than any transactions in the `txpool`. Due to the positive test result of t_4, Aleth's `txpool` would evict all transactions. In addition, the overdraft transactions are admitted and occupy the `txpool` until the locally found blocks consume them all.

12 DETER-Y: EXPLOIT FUTURE TXS AND RPC

12.1 Attack Design

Design motivation: As mentioned above, future transactions are free and incur no costs to the attacker. However, the challenge in designing low-cost attacks is how to make future transaction affect the pending queue in the `txpool`. We observe that if a future transaction is a local one (i.e., received on the victim node by RPC requests), the presence of a future local transaction would resize the `txpool`. Thus, we propose the second attack with zero cost, named DETER-Y. DETER-Y relies on the presence of RPC interface open on the victim node. In the following, we first describe the extended threat model in DETER-Y.

Extended threat model: We consider an extended threat model (based on § 3) where the victim node runs an RPC web service which accepts JSON requests and sends transactions on behalf of the EOA. Here, we assume that the attacker knows the victim RPC node; in § 13.2, we describe how the attacker can discover the victim RPC nodes in an operational Ethereum network.

Attack workflow: In the extended threat model, the attacker conducts the following steps:

Initially, the attacker obtained a conservatively guessed value of victim's `txpool` length n', say $n' = 10 \cdot 5120$.

y1 The attacker sends n' future transactions, at any Gas price,[10] through the RPC interface to the victim node. Note that here, the attacker only needs to send these n' local transactions for once.

y2 Before the next `txpool` read, the attacker node propagates to the victim node a future (non-local) transaction at an arbitrarily Gas price.

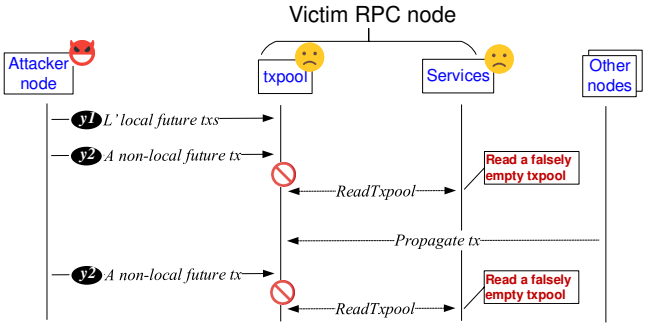


Figure 9: Workflow of a successful DETER-Y attack that sends one payload and resizes a victim `txpool` forever, so that the subsequent reads of the `txpool`, how many times it is, would return empty.

Attack analysis: In Step y1, the victim `txpool` receiving the n' future transactions would mark them local transactions. By default, the local transactions have priority in a `txpool` and can preempt other non-local transactions (i.e., propagated from other nodes), even when the local transactions have lower Gas prices than the non-local ones. In the case of n local transactions or more, the Geth `txpool` would maintain a single spot for non-local transactions, and allocate all other spots for local transactions. Note that local transactions are never dropped or evicted by the receiving RPC node, even there are more than n local transactions.

Thus, after Step y1, the `txpool` reduces its capacity to serve at most a single non-local transaction. As local future transactions cannot be executed by the miner, this reduces the capability of `txpool` to maintaining at most one executable transaction. Besides a miner, the `txpool` attacked by DETER-Y can affect other services. For instance, it can affect a RPC service and prevent a DApp

[10] These transactions are treated as local on the victim node. And a local transaction can always evict other types of transactions in the `txpool`, no matter what Gas price it is with.

client from viewing the pending transactions propagated from other nodes/client.

Note that the local future transactions are not evicted by the miner so they will occupy most slots in the `txpool`. Also, note that DETER-Y is extremely low cost, as the attacker EOA only needs to send n' transactions once. It does not need to repeatedly send transactions as the exploit of DETER-X does.

The single spot in the resized `txpool` is an FIFO queue. Thus, as long as the non-local future transaction in (y2) is sent close in time to when the `txpool` is read, the `txpool` would read the future transaction.

12.2 Local evaluation of DETER-Y

In our evaluation of DETER-Y, the attacker first invokes $n' = 5120$ `sendTransaction()` RPC calls ((y1)) and then propagates one future transaction for each (y2) message. We tested two rates, namely 0 future transactions per second (which means the single slot in the reduced `txpool` stores normal transactions) and 2 future transactions per second. During the 100-block period, the cumulative number of transactions included in the blockchain is reported in Figure 10b. This figure shows the disparity of the included transactions with and without DETER-Y. Compared with the DETER-X result of the same attack rate, DETER-Y causes lower number of transactions included, imply higher effectiveness/success rates. For instance, when the attack rate is 2 future transactions per second, DETER-Y results in a lower number of included normal transactions or a higher attack success rate at 93% than DETER-Y's success rate at 84.9%. When there is no (y2) transaction sent (at a zero rate), the attack success rate is 88.6%.

13 VICTIMIZE MORE TXPOOL SERVICES: STRATEGIES AND EVALUATION

Beyond miners and mining pools, DETER victims include other critical Ethereum services that depend on a `txpool`, including transaction propagation, RPC services, Gas station and other DApp services. In each victim, we descibe the attack strategies and evaluation results (on testnets).

13.1 Victimize Transaction Propagation

13.1.1 Attack Strategy. Preliminary on transaction propagation: An Ethereum node decides to propagate an incoming transaction only when the transaction passes precheck and is classified as a pending transaction. More specifically, in the Geth implementation, if the precheck emits a valid pending transaction, the node raises an event handler by a separate thread that scans the `txpool` to discover other propagate-able transactions that recently become pending. The thread then propagates all of them to the node's neighbors.

Analysis of DETER-X/Y/Z on propagation: The transaction-propagation thread is triggered upon the arrival of each valid pending transaction. In practice, if this arrival rate is high, it may render the existing attacks, DETER-X and DETER-Y, impractical – both attacks need sending attack messages at a high enough rate to frontrun the victim service's action of reading `txpool`. In this case, the attack rate needs to be higher than the arrival of incoming normal transactions.

By contrast, DETER-Z's crafted transactions can occupy the `txpool` which further fail the precheck of subsequent incoming transactions, thus able to deny the propagation of these subsequent transactions.

Strategy: An attacker interested in disabling the transaction propagation of a victim node mounts the DETER-Z attack.

13.1.2 Attack Evaluation in a Private Network. This experiment evaluates the effectiveness of the three DETER attacks on disabling the transaction propagation. The experiment runs in a private network of four nodes: A victim node, an attacker node that sends crafted transactions to the victim, an observer node that receives the transactions propagated from the victim as the target metric, and a normal node that sends normal transactions to the victim node. Thus, in this private network, the victim node is connected to the other three nodes, and there are no other connections. During each experiment, the normal node sends 2 transactions per second to the victim, and the process lasts for 250 seconds. We report the number of normal transactions received by the observer as the metric of attack effectiveness.

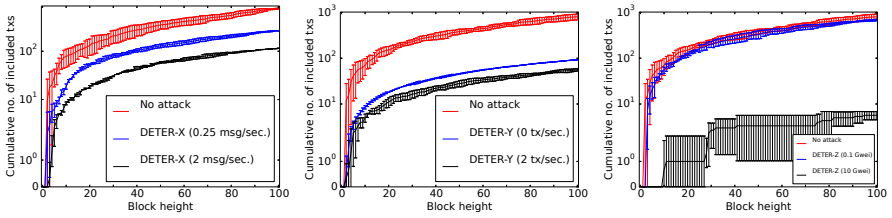
When evaluating DETER-X, the attacker sends each (x1) message with $n' = 5120$ crafted future transactions at a rate of 1 message per second. When evaluating DETER-Y, the attacker first invokes $n' = 5120$ `sendTransaction()` RPC calls ((y1)) and then propagates one future transaction ((y2)) per second to the victim. When evaluating DETER-Z, the attacker, upon receiving a new block, sends a message of $n' = 5120$ pending transactions ((z1)) and a message of $n' = 5120$ future transactions ((z2)), to the victim.

In each experiment, we report the cumulative number of propagated transactions from the 0th to 250th second. We run experiments for three times and report the average cumulative number with standard deviation in Figure 12a. The result shows that without attacks, there are 500 transactions successfully propagated by the victim at the end of experiment (the 250-th second). With DETER-X (DETER-Y), the number is 479 (490) transactions propagated, leading to an attack success rate of $1 - \frac{479}{500} = 4.2\%$ ($1 - \frac{490}{500} = 0.5\%$). Neither DETER-X nor DETER-Y is effective in disabling the transaction propagation of the victim. By comparison, with DETER-Z, the number is 62 transactions propagated, leading to an attack-success rate of $1 - \frac{62}{500} = 87.6\%$. DETER-Z is much more effective than the other variants in terms of attacking the transaction propagation, which validates our attack strategy.

For DETER-Z, we further vary the rates from 1 to 8 normal transactions per second and report the rate of propagated transactions (the inverse of success rate) in Figure 12b. The result shows that the propagation rate is insensitive and independent to the normal transaction rate (11.5% – 12.3%).

13.2 Victimize RPC Service

13.2.1 Attack Strategy. Preliminary: A RPC service is an off-chain service that facilitates a sender account to send transactions to the Ethereum blockchain. In practice, RPC services are widely used; it is estimated that the transactions sent through one RPC service alone (i.e., infura.io) accounts for at least 63% of all transactions on Ethereum [25].

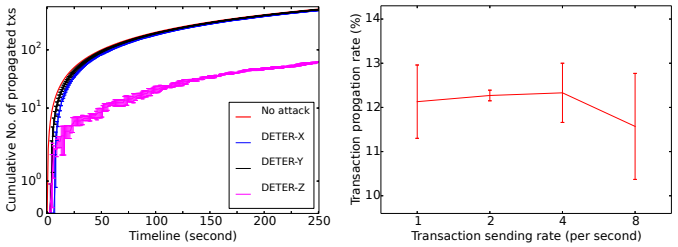


(a) Included txs under DETER-X (b) Included txs under DETER-Y (c) Included txs under DETER-Z

Figure 10: DETER a miner in a local network

DETER - attack rate	Success rate		Cost (Ether)
	Measured	Theoretical	
X-.25msg/sec	65.2±4%	61.33%	0
X-2msg/sec	84.9±3%	91.32%	0
Y-.25tx/sec	89.1±0.3%	N/A	0
Y-2tx/sec	93±0.7%	N/A	0
Z~ .25tx/sec	99.3±0.1%	100%	0.021/block
Baseline	96.6±0.1%	100%	12.5/block

Figure 11: DETER & baseline attacks



(a) Number of propagated transactions (b) DETER-Z w. varying normal tx rate

Figure 12: DETER transaction propagation

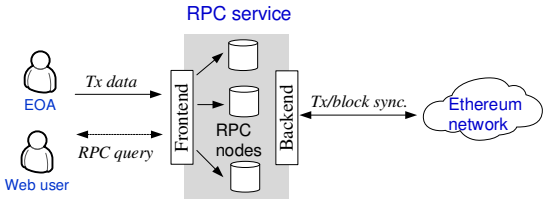


Figure 13: Model of an RPC service: It internally runs several RPC nodes in sync. with a remote Ethereum network. It supports two workflows: relaying transactions and processing RPC queries.

A RPC service receives a JSON-RPC request encoding a raw transaction from a sender account, wraps it in a transaction, and sends the transaction to the Ethereum nodes the service privately runs, which further propagate the transaction to the rest of the Ethereum network. In addition to this workflow of relaying transactions, a RPC service may also support querying the blockchain state (e.g., the set of transactions and smart-contract states like token balances). In particular, a RPC query `eth_getTransaction()` allows a web user to access both confirmed and unconfirmed transactions on the service nodes. The two workflows are depicted in Figure 13, where the service frontend is web servers receiving RPC requests and the service backend consists of a subset of Ethereum nodes under the service provider's control that are connected to the nodes in the rest of Ethereum network.

Threat model: An attacker has the capability of connecting to the nodes on the service backend and sending crafted messages to deny the RPC service in one of two workflows, that is, 1) to block or introduce significant delays to the confirmation of transactions sent through a RPC service, and 2) to cause a RPC service to return incorrect results on stale blockchain states. In this work, we focus DETER attacker's strategy on threat 1). There are alternative attack vectors for threat 2) [52].

Strategy to DETER RPC services: Given a victim RPC service, the attacker first identifies the nodes running on the service backend. In practice, this can be done in two steps: First, it discovers the client version of the RPC service (via `eth_clientVersion`) and the client version of any connected Ethereum node (via the handshaking with an Ethereum neighbor to be). Second, it matches the service client version with a node's version; the match implies that the node is run by the service provider.

Then, the attacker gets connected to the service-backend nodes, in a similar approach as in attacking top miners (described in § 6.2).

After that, the attacker is ready to mount the DETER attacks to the victim nodes. Here, because the attacker aims to delay the transaction propagation of backend nodes, he chooses DETER-Z (recall the strategy in § 13.1). Thus, the attacker's strategy is this: Upon each block received on the Ethereum network, the attacker sends two DETER-Z messages (z1 and z2) to all identified service-backend nodes in parallel.

13.2.2 Attack Evaluation with Testnets. **DETER-X on infura**: We choose the infura.io RPC service on the Goerli testnet as the evaluation subject. This choice is mainly due to that infura.io (on the mainnet) is the largest RPC service and Goerli is one of the top Ethereum testnets. In our experiment, we set up two Geth nodes in Goerli: An attacker node that connects only to the identified 11 service-backend nodes and a monitor node that randomly connects to 50 nodes in the testnet. We use the method described above to discover the service-backend node, by discovering infura's Geth version (code name *omnibus*) through sending `eth_clientVersion` RPC queries to the service frontend and conducting a one-day-long study to collect as many Goerli nodes' client version revealed in the handshaking subprotocol. During the one-day measurement study, we launch a Geth node statically instrumented to log all peer-discovery messages it received from the neighbor it does handshake with.

The target metric is the transaction propagation delay between when the transaction (more precisely the RPC request encoding the raw transaction data) is sent to the RPC service and when the transaction is observed by all nodes in the network. In the experiment, we use an account to send a pending transaction through the Goerli-infura's service frontend. We measure the time when the raw transaction is sent and the time when the transaction is received on the monitor node (which is statically instrumented to record the timing of all incoming transactions/messages).

Table 9: Transaction delay on infura RPC service in Goerli (w. DETER-Z)

Gas price of normal tx (Gwei)	Delay w.o. attacks (seconds)	Delay w. DETER-Z (seconds)
0.1	0.167 ±0.005	> 10, 800
1	0.153 ±0.06	> 10, 800
10	0.126 ±0.03	> 10, 800
100	0.119 ±0.05	> 10, 800
1000	0.128 ±0.04	0.155 ±0.07
2000	0.141 ±0.003	0.136 ±0.06

Following the above method, we conduct experiments to measure the delay without the attack: The experiments run ten times and we report the average delay with its standard deviation in Table 9. We then conduct experiments under DETER-Z attacks. In the attack, the crafted pending transactions' Gas price is configured at 1000 Gwei. With a fixed Gas price of the normal transaction, we run experiments for ten times and report the average delay and its standard deviation. We vary the Gas price of the normal transaction from 0.1 Gwei to 2000 Gwei, and report the result in Table 9.

The result shows that if the Gas price of the normal transaction sent via a RPC service is higher than the Gas price of the DETER-Z attack, there is not much delay introduced by the attack. Otherwise, that is, with DETER-Z's Gas price higher than the normal transaction's, the delay of the normal transaction grows to be at least longer than 3 hours (10800 seconds as we did in the experiment)[11]. This result is particularly interesting, and implies that applying DETER-Z to a RPC service cannot only delay but also block the propagation of a transaction. The reason is that with *all* service-backend nodes being DETER-Z attacked, the normal transaction does not have alternative paths to reach the rest of the Ethereum network. At the time when the DETER-Z transactions are evicted after the next block is received, the Ethereum nodes inside the RPC service are done their attempts to propagate the normal transaction, thus leaving it discarded in the blockchain forever. Note that the result differs from the previous case of DETER-Z attacking a single node's propagation in a network, in which the normal transaction can still be propagated through alternative paths bypass the victim node to reach the entire network.

DETER-Y on blockdeamon: We evaluate the impact of DETER-Y on the RPC services that support local transactions. Although Infura does not support local transactions, there are other RPC services that support it, Blockdeamon is one of them. We measured that the txpool size of Blockdeamon's RPC node is 5120 in the Goerli testnet. Therefore, we send 5120 local future transactions through Blockdeamon's front-end in the Goerli testnet. We then use an EOA to send a pending transaction from our local node that connected to Goerli testnet. After sending the transaction, we keep retrieving it through Blockdeamon's front-end. We vary the Gas price when send the pending transaction and report the result in table 10. The result shows that when the EOA bids a Gas price that equals to or lower than 0.2 Gwei in the pending transaction, this transaction can never be retrieved from Blockdeamon's front-end (we investigated the observation and found that the Goerli testnet miners only include transactions that bid a Gas price higher than 0.2 Gwei), when the EOA bids a Gas price higher than 0.2 Gwei (from 0.4 Gwei to 6.4 Gwei), the retrieval delay is smaller (from 44 seconds to 5.4 second).

[11] 3 hours is the longest time that an unconfirmed transaction is stored on Geth's `txpool` before it is forced to be evicted.

Table 10: DETER-Y transaction retrieval on Blockdeamon

Gas bid (Gwei)	Retrieval delay (No attack)	Retrieval delay (DETER-Y)
0.1	20.39 ±3.86 ms	>10800
0.2	15.27 ±4.47 ms	>10800
0.4	51.84 ±27.47 ms	43990.0 ms
0.8	38.09 ±41.06 ms	26030.0 ms
1.6	14.64 ±4.64 ms	10200.0 ms
3.2	14.88 ±0.93 ms	7540.0 ms
6.4	15.56 ±10.25 ms	5400.0 ms

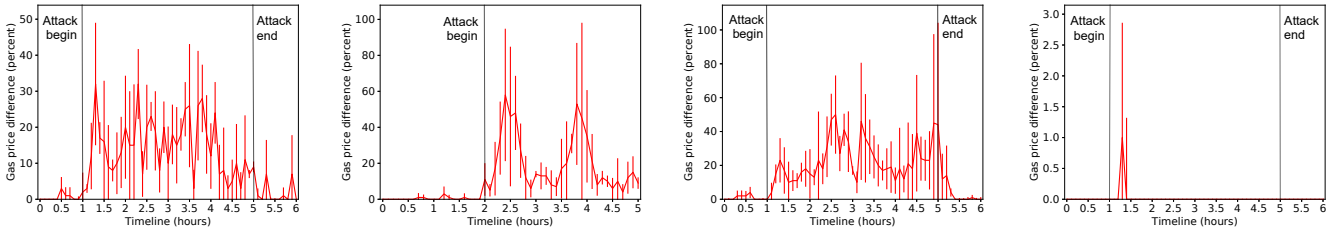
13.3 Victimize Gas Station Services

This experiment evaluates the DETER attacks' effectiveness on victim Gas station services. In the experiment setup, we launch two Geth nodes on two identical machines and join them to the mainnet. There is no direct connection between the two nodes. We run the EthGasStation code [30] on the victim and normal nodes. In our experiment, one node is used as the victim and the other is the normal node for comparison purposes. We also mount an attacker node in connection to the victim node and in synchronization with the mainnet.

In each experiment, we turn on the Gas station simultaneously on the two nodes. After a certain period of time (e.g., one hour), the attacker node sends payloads to the victim node with the following profiles: the attack rate is one message per second for DETER-X, 0 per second for DETER-Y (no $y2$ message is sent), and two messages per block for DETER-Z. For each attack, we run a process that lasts for 6 hours, during which the predicated Gas prices on both victim and normal nodes are collected every 6 minutes. Note that EthGasStation produces four different kinds of predicted Gas prices, codenamed by "Fastest"/"Fast"/"Standard"/"Safe Low," which respectively mean the predicted Gas price for the transaction to be included in the next block/within two minutes/within five minutes/within thirty minutes. We show the most representative result on "Standard" and "Fastest" Gas prices here.

The result of DETER-X on the Gas station is in Figure 14a. In the 6-hour experiment period, the attack started from the second hour and stopped in the end of fifth hour. When the attack begins, the difference of predicted prices between the victim Gas station and the normal Gas station starts increasing. The percentage of the difference (difference/normal price) reaches 10% after 10 minutes and varies from 10% to 40% as the attack goes on. The difference returns to less than 5% after the attack ends. Since the design of the Gas station considers history information to build the model and make the prediction, it takes time for the difference to return to less than 5%.

The result of DETER-Y on the Gas station is in Figure 14b. In the 6-hour experiment period, the attack started from the third hour and stopped in the end of sixth hour. The DETER-Y caused more damage to the Gas station than the DETER-X, with an average of 30% difference during the 3 hours. Since the only way to stop the attack is to restart the node and delete the transaction data, we didn't test how the Gas station recovers after the attack ends.



(a) Standard, DETER-X (b) Standard, DETER-Y (c) Standard, DETER-Z (d) Fastest, DETER-Z

Figure 14: The difference of predicted price between the victim Gas station and normal Gas station.

The result of DETER-Z on the Gas station is in Figure 14c. In the 6-hour experiment period, the attack started from the second hour and stopped in the end of fifth hour. The difference caused by DETER-Z is as great as that by DETER-Y. When the attack ends, it takes about 30 minutes for the victim Gas station to recover to normal.

13.4 More Results on Victimizing Mining Pools

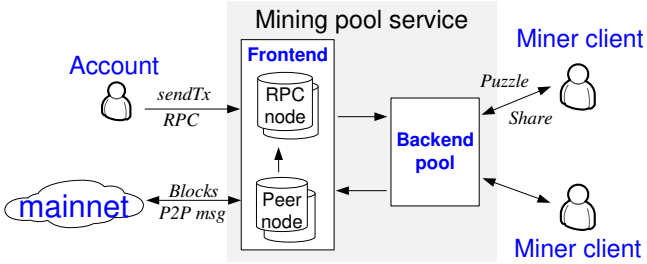


Figure 15: Mining pools in mainnet

Preliminary: Mining pools: In the core of a mining pool, the pool sends to each participating miner a easier version of the mining puzzle and receives from miners the solutions of the easier puzzles (so-called shares) before paying the reward out to the miners. Today, popular mining pools in practice grow into a complex, multi-layer ecosystem, depicted in Figure 15, with augmented interfaces on the frontend: 1) A RPC service interface to accept transactions submitted from its transaction clients, and 2) a block-receiving P2P interface through which the backend pool allows to be connected as client neighbors and propagates recently found valid blocks to the clients. These two interfaces are designed to facilitate the transaction and block propagation relevant to the mining pool. For instance, the most successful mining pool in Ethereum, namely Sparkpool, has a frontend service network named Taichi network, through which the backend pool receives transactions [35] and propagates blocks [34].

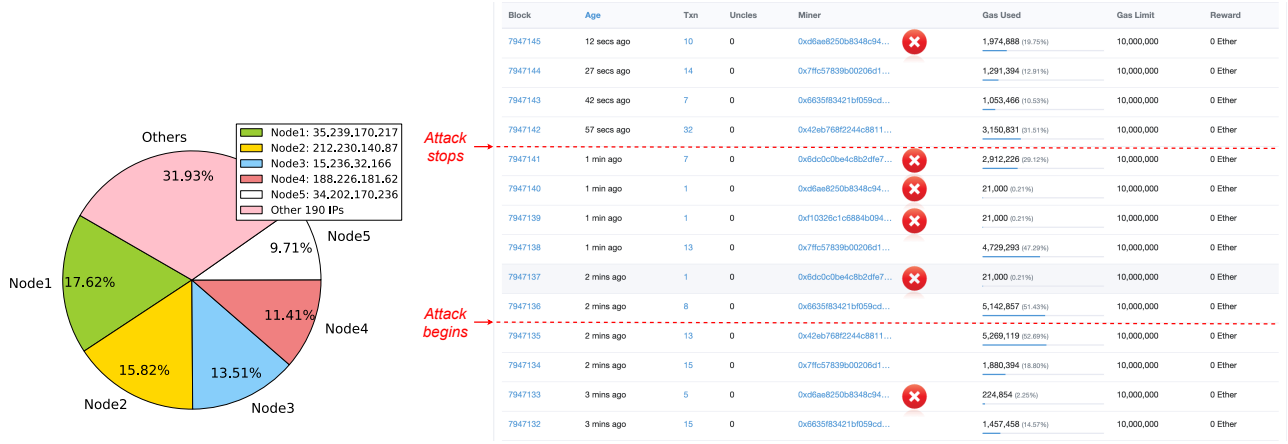
DETER-Z on Ropsten & Rinkeby: In this attack, the attacker node sends a message of $L' = 5120$ pending transactions (z1) and a message of $L' = 5120$ future transactions (z2), upon the recipient of each block. In each round, the two messages are sent to the selected miners in parallel. Due to ethical concerns, we configure the attack to last for 90 seconds (or 6 blocks) in Rinkeby and 120 seconds (or 10 blocks) in Ropsten.

The evaluation result on Rinkeby is presented in Figure 16b. The result shows, before the attack starts (blocks #7947132-7947135), around 5-15 transactions are included in each block and the consumed block Gas is between 2.2% and 52.7%. During the attack (blocks #7947136-7947141), 3 blocks (#7947137, #7947139, #7947140) only include 1 transaction and only 0.2% block Gas is used, we observed that it is our attack transactions were included in these blocks. The other 3 blocks (#7947136, #7947138, #7947141) include more than 1 transactions and 29.1%-51.4% block Gas is used. These 3 blocks were mined by a miner which is not selected as the victim. In summary, the result shows our DETER-Z attack can preempt a block by one of our attack transactions and exclude the inclusion of other regular transactions. The cost of DETER-Z is 0.126 Ether (0.021 * 6).

We report the evaluation result on Ropsten in Figure 17a. As can be seen from the result, before the attack starts (blocks #9511168-9511170), around 34 transactions are included at each block and more than 31% of block Gas is consumed. During the attack (blocks #9511171-9511180), 8 out of 10 blocks are preempted by one of our attack transactions and the block Gas usage is below 0.26%. Our attack failed at two blocks (blocks #9511173, #9511177) which were mined by the miners that not under our attack, with one block includes 0 transaction and the other block includes 79 transactions.

In summary, the evaluation results on the testnets show that the attack transactions sent in DETER-Z can preempt 3 out 6 (8 out of 10) blocks in Rinkeby (Ropsten) and prevent other regular transactions from being included into these affected blocks, leading to a 50% (80%) success rate.

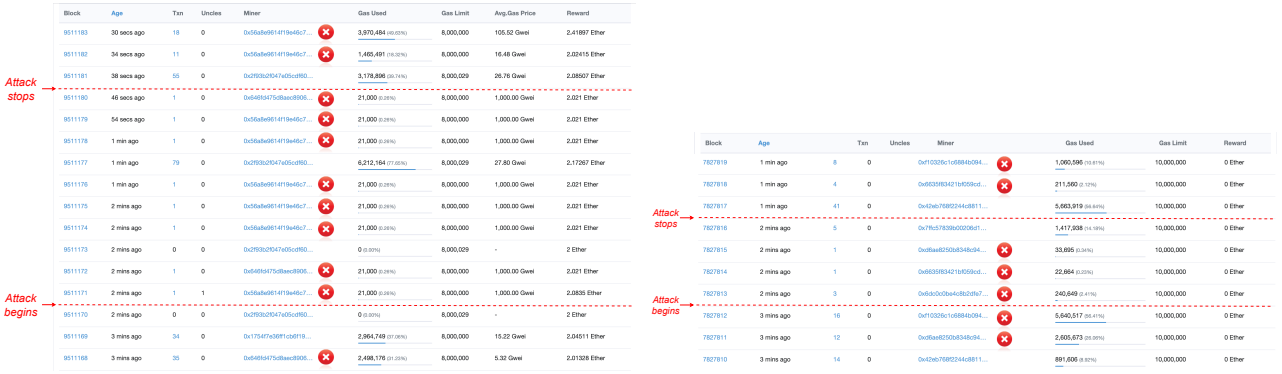
DETER-X on Rinkeby: Figure 17b presents the evaluation result of DETER-X on Rinkeby. We send $L' = 5120$ future transactions with a Gas price of 1000 Gwei at the rate of 1 message per second. The attack payload is sent to the selected top 5 miners in parallel and lasts for 60 seconds (or 4 blocks). As the figure shows, before the attack starts (blocks #7827810-7827812), around 12-16 transactions are included at each block and more than 8.9%-56.4% block Gas is used. During the attack, 3 out of 4 blocks are mined by the victim miners that under our attack, leading to a 75% success rate. In these 3 blocks, less than 3 transactions are included at each block and the block Gas usage is less than 2.4%. Our attack failed at one block (#7827816) which includes 5 transactions. The evaluation result on Rinkeby shows that our DETER-X can reduce the regular transaction inclusion rate by $4 - 12\times$ and reduce the block Gas usage by $4 - 200\times$.



(a) Top miners in Rinkeby

(b) Screenshot from etherscan.io

Figure 16: DETER-Z top-5 miners in Rinkeby



(a) Screenshot from etherscan.io (DETER-Z top-4 miners in Ropsten)

(b) Screenshot from etherscan.io (DETER-X top-5 miners in Rinkeby)

Figure 17: DETER-X/Z attacks on miners in testnets

13.5 Result Summary

Beyond the miners/mining pools described in the maintext, other critical Ethereum services that depend on the `txpool` can be DETER victims, such as mining pools, transaction propagation, RPC service, Gas station and other DApp services. We summarize the attack strategies and evaluation results to victimize these additional Ethereum services: 1) To block a node from propagating incoming transactions, the attacker mounts DETER-Z attack. A local-node evaluation shows that DETER-Z can prevents 87.6% of transactions from being propagated. 2) A RPC service provides transaction relay services for end users (the background on RPC will be described in § 2). By discovering nodes on the service backend and sending them DETER-Z payloads, one can block transactions replayed through a RPC service. Evaluation shows that DETER-Z on infura' s nodes on the Goerli testnet can infinitely delay the relay/propagation of transactions of Gas price lower than 1000 Gwei. 3) Gas station is a service that reads from `txpool` and ledger the transactions' Gas prices to suggest a proper value of Gas price for incoming transactions. Mounting DETER-X/Z on an Ethereum node running the popular EthGasStation code [29], one can alter the predicted Gas price by as much as 50%/100%.

14 MITIGATION SCHEME M_0 AND EVALUATION SETUP

14.1 Mitigation Scheme M_0

Table 11: Example of authorized searcher attack

Policy	Admission	Eviction
M_0	No admission of future transaction	NA
	No admission of invalid transaction	
	No admission of tx of the same sender with any txs in the pool	
M_1	NA	No eviction of valid tx by future/invalid tx
		No eviction of valid tx that transforms existing valid tx into future or invalid.

M_0 adds three transaction-admission rules to an underlying `txpool`: M_{0a}) It does not admit any future transitions. M_{0b}) It does not admit any invalid transaction. M_{0c}) It does not admit a transaction that shares the same sender with another transaction currently residing in the `txpool`.

Security analysis: A `txpool` under admission policy M_0 is secure against DETER attacks (that is, any DETER attack fails). Because of Theorem 14.1, there is no future or invalid transaction in the `txpool`. A DETER attack that relies on populating future or invalid transactions (respectively in DETER-X or DETER-Z) into the `txpool` cannot succeed.

Theorem 14.1. *Under admission policies M_0, at any time, the* `txpool` *does not contain any future transaction or an overdraft transaction.*

Proof. We first prove that under admission policy M_0, at any time, the `txpool` does not contain any future transaction. If at a certain time, there is a future transaction in `txpool` under M_0, the future transaction must either be directly admitted to the `txpool` or is admitted as a pending transaction and then is transformed to a future transaction by eviction. The former case is directly prohibited by the admission policy in M_0. The latter case is also impossible because to transform a transaction tx_2 to a future one, its predecessor transaction tx_1, that is the transaction of the same sender with tx_2 but having the nonce smaller than that of tx_2 by one, needs to be evicted by the `txpool`. However, M_0 prohibits two transactions of the same sender, like tx_1 and tx_2, in the `txpool`. Thus, the latter case does not hold.

We then prove that under admission policy M_0, at any time, the `txpool` does not contain any overdraft transaction. Because M_0 prevents admitting overdraft transactions, if there has to be an overdraft transaction in the `txpool`, it has to be transformed. That is, such a transaction, say tx_1, must not be an overdraft at the time of admission and after being admitted to the `txpool`, it is transformed into an overdraft by the inclusion/admission of another transaction, say tx_2, that spends the tx_1's balance. However, to do so, tx_2 must be from the same sender with tx_1, which violates M_0. □

Cost analysis: While scheme M_0 is sufficient to mitigate DETER attacks, it is not necessary. And this can cause excessive loss of miners' revenue. For instance, M_0 prevents any transactions of the same sender being admitted. However, it is possible that two transactions of the same sender are valid transactions, and M_0 rejecting them clearly makes the `txpool` lose the chance to collect their fees. Besides, M_0 prohibits any future or invalid transactions upon admission, which is another overkill. The transaction state upon admission is temporary and can be changed. For instance, a future transaction at the time of admission, say tx_2 of nonce $n+2$, can be transformed into a pending transaction if another transaction tx_1 of nonce $n+1$ arrives at the same `txpool`. Thus declining such a temporary future transaction leads to miners' inability to collect its transaction fee.

14.2 Evaluation Setup

14.2.1 Experiment Methodology & Setup. **Macrobenchmark by real transactions**: To construct the macrobenchmark, we launch a Geth node joining the mainnet in order to collect *any* transactions that arrive at the node, no matter what transaction states they are in. To do so, we statically instrumented the Geth client to intercept arriving transactions prior to any admission checks. After the launch of the Geth node, we wait three hours for the synchronization process to finish. We then start to 1) take the snapshot of the `txpool` by recording all currently residing transactions, and 2) collect the received transactions in the next one hour. In the one-hour period, we recorded a total of more than five hundred thousand transaction.

In order to replay the collected transactions (as will be described), we also record the amount of Gas each transaction consumes. This is done by observing the produced block in the mainnet and record the Gas of the collected transactions. It is possible that certain transactions collected are not included in the mainnet block. For this kind of transactions, we just use their Gas limit (i.e., the maximal amount of Gas allowed by the transaction sender) to simulate its (otherwise) used Gas.

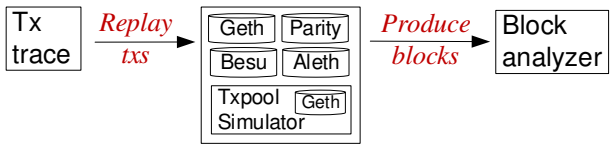


Figure 18: Experiment platform: Transaction replay and `txpool` simulation

Revenue measurement by replaying transactions: To evaluate the miners' revenue collected by a client, we employ the approach of replaying transactions. That is, we drive the collected transactions to a running client initialized with the initial `txpool` content, and turn on the mining to observe the produced blocks and the transactions included there. Technically, we replay each transaction in the benchmark with the following modifications: 1) We replace the transaction's public key by the one generated by us, so that we can know the private key and sign the modified transaction. 2) Given a transaction that invokes a smart contract, we modify the transaction's `data` field to make it invokes a designated smart contract that we call GasSimulator. The function in GasSimulator takes as argument a value g and runs a loop of SHA3 instructions such that the function call actually consumes approximately g Gas. The modified transaction calls GasSimulator with argument g where g is the amount of Gas consumed in the benchmark (e.g., in the real Ethereum transaction as in macrobenchmark). The transaction-replay engine and evaluation system are illustrated in Figure 18.

For fair transaction replay, our goal is to ensure that each mining event occurs at the same position in the sequence of replayed transactions on different Ethereum clients. For instance, the proof-of-work mining event that selects transactions from `txpool` right after the i-th replayed transaction and removes the selected transactions right after the j-th replayed transaction does the same on all different tested Ethereum clients (or our proposed mitigation schemes). Ensuring this property is non-trivial, as different Ethereum clients produce blocks at different rates even under the same mining difficulty. Instead of tuning difficulty directly (which we found is hard to control), we empirically tune the rate in which transactions are sent for replaying, so that the rate of transactions sent matches the rate of mining on specific miners. With this, we ensure each client produces the same number of blocks when replaying the collected transactions in a benchmark (e.g., 248 blocks under the macrobenchmark).

Implementing mitigation schemes: We implement mitigation schemes in a `txpool` simulator. Specifically, the simulator receives transactions replayed from the benchmark and periodically pushes transactions to a downstream Geth node for mining. The `txpool` simulator implements a simple unsorted array to store transactions. When the downstream Geth node is about to mine, the simulator selects a number of pending transactions, ordered by

their Gas prices, whose total Gas is twice the Ethereum block limit. Then the simulator sends this batch of transactions to the Geth node which runs mining and produces one block. Upon observing a newly produced block, the simulator pauses the mining, deletes the transactions included in the block and then requests the Geth to clear its own `txpool`. The simulator implements basic functionality such as transaction replacement (i.e., replacing an existing transaction by an incoming transaction of the same sender but at higher Gas price).

On top of the `txpool` simulator, we implement the two mitigation schemes M_0 and M_1 by customizing the operations for transaction admission (M_0) and eviction (M_1). Particularly, the eviction policy in M_1 enforces the following priority, that is, $VH > VL > [FI]H > [FI]L$.

14.3 Additional Testnet Measurement Results

Geth percentage in Testnets: We also measure the versions of Ethereum client software running on all nodes in the three testnet. The purpose is to understand how many nodes run Geth in these Ethereum testnets, as our DETER is designed mainly for Geth clients. To do so, we run a measurement Geth node, for about three days in Oct, 2020 (three days for each one of the three testnets), to passively let it connect to all nodes in a target testnet, say Ropsten. Node M can know the client software running on its neighbor because when establishing node connections, the Ethereum subprotocol's handshake phase requires the two nodes to exchange client version information [51].

Table 12: Percentage of Geth nodes in three testnets

Testnet	Geth	Parity	Others
Ropsten	92.5%	6.2%	1.3%
Rinkeby	98.7%	1%	0.3%
Goerli	94.2%	3.5%	2%

The results are in Table 12. For all three major testnets, most nodes there run Geth client, that is, 92.5% for Ropsten, 98.7% for Rinkeby and 94.2% for Goerli. By contrast, there are fewer than 7% nodes running Parity[12], the second most popular Ethereum clients. With this, we believe measuring the edges among Geth clients capture most edges in an Ethereum testnet.

[12] After Parity 3.0, the software is renamed to OpenEthereum. In this paper, we still use "Parity" to refer to both Parity with versions older than 3.0 and OpenEthereum.