No Fish Is Too Big for Flash Boys! Frontrunning on DAG-based Blockchains

Jianting Zhang* Aniket Kate*†
*Purdue University, †Supra Research

***Abstract*—Frontrunning is rampant in blockchain ecosystems, yielding attackers profits that have already soared into several million. Most existing frontrunning attacks focus on manipulating transaction order (namely, prioritizing attackers' transactions before victims' transactions) *within* a block. However, for the emerging directed acyclic graph (DAG)-based blockchains, these intra-block frontrunning attacks may not fully reveal the frontrunning vulnerabilities as they introduce block ordering rules to order transactions belonging to distinct blocks.**

This work performs the first in-depth analysis of frontrunning attacks toward DAG-based blockchains. We observe that the current block ordering rule is vulnerable to a novel *inter-block* frontrunning attack, which enables the attacker to prioritize ordering its transactions before the victim transactions across blocks. We introduce three attacking strategies: *(i)* Fissure attack, where attackers render the victim transactions ordered later by disconnecting the victim's blocks. *(ii)* Speculative attack, where attackers speculatively construct order-priority blocks. *(iii)* Sluggish attack, where attackers deliberately create low-round blocks assigned a higher ordering priority by the block ordering rule.

We implement our attacks on two open-source DAG-based blockchains, Bullshark and Tusk. We extensively evaluate our attacks in geo-distributed AWS and local environments by running up to $n = 100$ nodes. Our experiments show remarkable attack effectiveness. For instance, with the speculative attack, the attackers can achieve a 92.86% attack success rate (ASR) on Bullshark and an 86.27% ASR on Tusk. Using the fissure attack, the attackers can achieve a 94.81% ASR on Bullshark and an 87.31% ASR on Tusk.

We also discuss potential countermeasures for the proposed attack, such as ordering blocks randomly and reordering transactions globally based on transaction fees. However, we find that they either compromise the performance of the system or make the protocol more vulnerable to frontrunning using the existing frontrunning strategies.

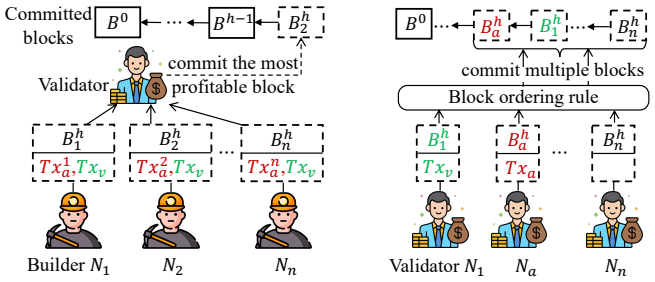
1. Introduction

Blockchain technology is revolutionizing the regulated financial market via powerful decentralized finance (DeFi) protocols, which implement a transparent and decentralized market for users to perform economic activities, e.g., cryptocurrency trading. As of September 2024, over 85 billion USD worth of cryptocurrencies has been staked in the mainstream DeFi platforms [1]. Unfortunately, the booming market fosters illegal behaviors: frontrunning is rampant. In the blockchain context, the frontrunning attack allows the adversary to prioritize its transactions before the victims' transactions. As transactions can impact the states of DeFi, e.g., buying a cryptocurrency will increase its price, such order-prioritized frontrunning is profitable and has shown to be happening at a massive scale [2]–[12]. For instance, a frontrunning called sandwich attack has yielded 41 million USD since Ethereum was merged [13]. Moreover, the frontrunning attackers are estimated to extract millions of USD with varying frontrunning attacks, such as suppression attack [4], imitation attack [10], and arbitrage across Rollups [12]. Besides monetary losses of DeFi users, frontrunning is proven to threaten consensus stability and aggravate network congestion [3], [6], [14].

Despite extensive explorations and studies on frontrunning attacks, existing efforts mainly focus on the Ethereum network [15]. Therefore, the existing exploited frontrunning strategies are subjective to Ethereum's transaction ordering rule, which determines how transactions are ordered *within a block*. Nodes in the current Ethereum network order transactions based on the maximal extractable value (MEV), which is empowered by a proposer-builder separation (PBS) scheme [16]. Figure 1a demonstrates the MEV-based ordering rule. The builders first create distinct profitable blocks with specific transaction orders, such as prioritizing their transaction Tx_a^i before the other's transaction Tx_v or ordering transactions based on the receivable transaction fee. The proposer/validator then selects the most profitable block (i.e., the maximal extractable value it can earn) and commits the block into the blockchain[1]. Under the MEV-based ordering rule, attackers can frontrun the victim transactions using various attacking strategies, such as increasing transaction fees by engaging in priority gas auctions [3] or acting as a builder or validator to customize the order. However, once blockchains employ distinct ordering rules, these frontrunning strategies will become ineffective.

We observe that the MEV-based ordering rule is not unique to the gradually growing blockchain markets. A family of the directed acyclic graph (DAG)-based blockchains have been recently proposed to improve the transaction throughput of the system and experienced rapid growth [17]–[26]. For instance, Sui [27], a production DAG-

1. Note that the PBS scheme involves multiple roles in ordering and committing blocks, including searchers, builders, relays and proposers. We simplified the scheme here for illustration purposes.



(a) Transaction ordering in the Ethereum network

(b) Transaction ordering in DAG-based blockchains

Figure 1: Frontrunning attacks on distinct blockchains, where an attacking transaction Tx_a is intentionally ordered before a victim transaction Tx_v. (a) Frontrunning happens within a block against the MEV-based ordering rule in Ethereum. (b) Frontrunning happens across blocks against the block ordering rule in DAG-based blockchains.

based blockchain with a market cap exceeding 4 billion USD and powering over 100 DeFi protocols as of September 2024 [28], can achieve over 400K transactions per second (TPS) of raw transactions according to their latest technical report [21]. Besides, Aptos [29] and Celo [30] are also developing their DAG-based consensus protocols. Unlike the single-chain-based ledgers (e.g., Ethereum), where validators can only order and commit a block of transactions each time, the DAG-based blockchains enable validators to *order multiple blocks* within a single instance of consensus. Specifically, a DAG-based blockchain processes transactions round-by-round. In each round, every node (i.e., validator) can propose a block including transactions and connections to blocks of the previous round. All the proposed blocks and connections are then disseminated to form a DAG. Due to the encoded information in the DAG, once a block is committed via an instance of consensus, all its causal history (i.e., all blocks for which there is a connection or path from the committed block to them) can also be ordered and committed. Consequently, to globally order transactions, the DAG-based blockchain requires each validator to not only order transactions within a block but also order blocks in the causal history via a *block ordering rule*. Due to these new ordering features, the existing frontrunning strategies that target the MEV-based ordering rule may no longer be effective. This makes us wonder:

Are the DAG-based blockchains more resilient to frontrunning attacks?

It is non-trivial to answer this question. On the one hand, the attacker has limited spaces in frontrunning transactions within a block (which are called *intra-block frontrunning* in this paper). Specifically, the DAG-based blockchains allow validators to create blocks simultaneously. To avoid duplicate transactions packed in multiple blocks, each transaction is only assigned to one validator for ordering and commitment. In this case, the attacker cannot intra-block frontrun the victim transaction if it is not assigned to package the victim transaction. Intra-block frontrunning becomes less likely if the number of validators is large, and victim transactions are expected to be assigned to validators randomly (in which the adversary cannot pick up victim transactions arbitrarily). On the other hand, the DAG-based blockchain introduces a block ordering rule to order transactions of causal history blocks. The new ordering rule may expose DAG-based blockchains to new frontrunning vulnerabilities.

In this paper, we provide the first in-depth analysis of frontrunning attacks toward DAG-based blockchains. We discover that the widely adopted block ordering rule in DAG-based blockchains is susceptible to a novel *inter-block frontrunning attack*. Unlike the intra-block frontrunning working on directly manipulating the order of transactions within a block, the inter-block frontrunning attack allows the attacker to intentionally order its blocks before the victims' blocks, and consequently, the transactions in the attacker's blocks will be ordered before the transactions in the victims' blocks. Figure 1b presents a brief process of the inter-block frontrunning attack. Specifically, a frontrunning attacker N_a constructs a block B_a^h containing an attacking transaction Tx_a to frontrun a victim transaction Tx_v in another block B_1^h. By employing our proposed attacking strategies (Section 4), the attacker N_a can ultimately ensure that B_a^h is ordered before B_1^h, resulting in Tx_a frontrunning Tx_v. This paper explores the unreasonable designs of the vulnerable block ordering rule that spawns the inter-block frontrunning strategies, evaluates the attack effectiveness under different attacking strategies, and discloses factors that impact the attack effectiveness.

Our contributions can be summarized as follows:

- We present and formulate the first inter-block frontrunning attack against the widely employed block ordering rule in DAG-based blockchains (Section 3).
- We provide insightful analysis of the order vulnerabilities in DAG-based blockchains (Section 4). Our investigation reveals ordering priorities, which motivates us to explore three attacking strategies, including the fissure attack (Section 4.2), the speculative attack (Section 4.3), and the sluggish attack (Section 4.4). We demonstrate how attackers can utilize these attacking strategies to improve the attack success rate (ASR).
- We implement our frontrunning strategies on two open-source DAG-based blockchains (i.e., Tusk [18] and Bullshark [19]) and conduct extensive experiments to evaluate them in the geo-distributed and local environments (Section 5). The experiment results show remarkable attack effectiveness. For instance, using the speculative attack on the AWS environment, the attacker can achieve ASRs of 85.19% to 92.98% on Bullshark and 81.08% to 87.76% on Tusk under varying numbers of validators. Using the fissure attack, the attacker can achieve ASRs of 92.74% to 94.81% ASRs on Bullshark and 82.56% to 87.31% on Tusk under varying numbers of attackers.
- We discuss potential countermeasures (Section 6), yet, none can be seamlessly integrated into the existing DAG-based blockchains without compromising performance or introducing new frontrunning vulnerabilities.

To this end, we generalize and model our explored attacking strategies (Appendix B), which disclose factors impacting the attack effectiveness. We hope such generalized models can provide insights for future researchers on designing effective mitigations.

Responsible Disclosure. We initially shared and discussed our findings about MEV and frontrunning issues on DAG blockchains with Mysten Labs's team working on Sui on June 5, 2024. We then provided them with a detailed paper on September 10, 2024. They have acknowledged the proposed attacks. They plan to explore long-term mitigation to handle these attacks.

2. Preliminaries and Related Work

This section first introduces the preliminaries of DAG-based blockchains. Then, we present existing frontrunning attacks and relevant mitigations, followed by our motivation.

2.1. DAG-based SMR

Blockchains rely on a fault-tolerant state machine replication (SMR) process to maintain a transaction ledger[2]. In an SMR protocol, n nodes $\{N_1, \cdots, N_n\}$ (of which up to t can be malicious) maintain an ever-growing sequence of transactions by repeatedly performing three tasks: data dissemination, ordering, and execution. The SMR protocol can guarantee the following intrinsic security properties:

Definition 1 (Safety)**.** *If sequences of transactions* $(tx_1, tx_2, \cdots, tx_j)$ *and* $(tx'_1, tx'_2, \cdots, tx'_{j'})$ *are committed by two honest nodes, then* $tx_i = tx'_i$ *for all* $i \leq min\{j, j'\}$.

Definition 2 (Liveness)**.** *If a transaction* tx *is sent to at least one honest node, then* tx *will be eventually committed by every honest node.*

A DAG-based SMR protocol carries on the safety and liveness guarantees while incredibly enhancing the system throughput by decoupling data dissemination from metadata ordering. Briefly, in the network communication layer, nodes disseminate transactions and construct blocks of transactions to form a directed acyclic graph (Section 2.1.1). With the constructed DAG, the consensus logic then introduces zero communication overhead to order and commit transactions by employing a block ordering rule (Section 2.1.2).

2.1.1. Construction of a DAG. A DAG-based SMR processes in *rounds* $\{r_1, r_2, \cdots\}$. In each round, every node creates one new block that connects blocks of the previous round. The round-by-round blocks and their connections eventually form an ever-growing DAG, where blocks serve as vertices and connections between blocks serve as edges. Furthermore, blocks and connections piggyback the following information to guarantee safety and liveness.

- **Blocks**: A block B_k^i created by node N_k in round r_i consists of a collection of transactions (or transaction batches) and at least $n - t$ connections to blocks from the previous round r_{i-1}. Transactions (or transaction batches) in the block are locally ordered.
- **Connections**: A block B_k^i includes a set of connections that piggyback its *local ordering preference (LOP)* on its connected blocks. The LOP defines a local order for the connected blocks and will be used to establish a consistently global order for all blocks of the DAG.

The constructed DAG defines partial orders between transactions within a block and between the connected blocks. However, the DAG is not fully connected, and there are some missing connections between blocks. To establish a global order, nodes need to interpret the DAG to decide which blocks can be eventually ordered and committed. In a DAG-based SMR, nodes interpret the DAG by dividing it into *waves*. Each wave consists of several consecutive rounds[3] and works to select a unanimous *leader block* (also called *anchor*) for the last wave [20]. Once an anchor is decided, all its causal history (i.e., all blocks for which there is a connection and path from the anchor to them) will be ordered and committed (see Section 2.1.2 below).

Figure 2a presents a DAG-based SMR maintained by $n = 4$ nodes. The DAG is structured by three-round waves, and there is an anchor in each wave deciding blocks that can be committed. For instance, the wave w_{j+1} selects an anchor B_2^{i+2} in round r_{i+2}, and then all the historically uncommitted blocks of B_2^{i+2} (highlighted by green color) are eventually committed.

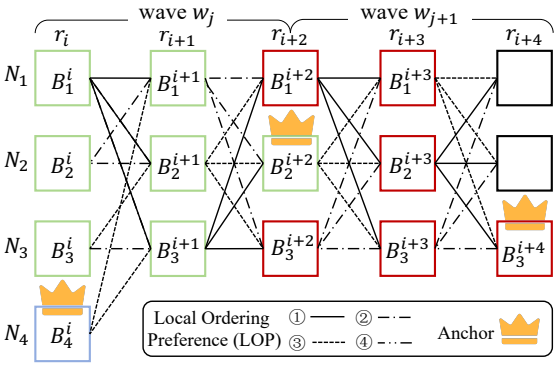
2.1.2. Ordering transactions in DAG-based SMR. With a DAG of blocks formed in the network communication layer, nodes then establish a consistently global order for transactions in the consensus layer. As each block indicates a local order of its packed transactions, the task performed by the consensus layer is to order blocks. In particular, the DAG-based SMR orders blocks via the following steps [20]:

1) *Decide anchors*. Nodes select an anchor for each wave. The anchors can be pre-determined in partially synchronous DAG protocols (e.g., Narwhal-Hotstuff [18] and partially synchronous Bullshark [31]), or randomly decided by a specific round of the wave in the asynchronous DAG protocols (e.g., DAG-Rider [17], Tusk [18], and asynchronous Bullshark [19]).
2) *Order anchors*. Some anchors may have been decided but not committed due to network delay or malicious behaviors. Therefore, once a new anchor is decided, nodes trace the uncommitted anchors from previous waves. All these uncommitted anchors are eventually ordered based on their wave numbers.
3) *Order historical blocks*. Once the uncommitted anchors are ordered, nodes trace the causal history of each of them. Thanks to the encoded information in the DAG, every node can obtain the same historical blocks[4].

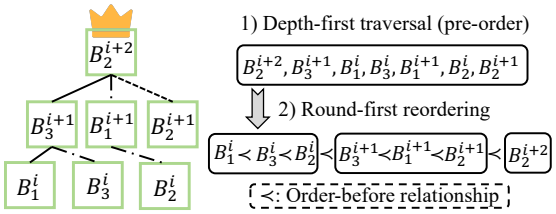
2. In this paper, we use the terms 'blockchain' and 'SMR' interchangeably as they hold the same meaning in the DAG-based blockchain context.

3. A wave in different DAG-based SMR protocols could be defined to contain different numbers of rounds for, e.g., latency optimization.

4. We omit the security proof here but recommend that interested readers read the proof provided by the original papers [17]–[19].



(a) The structure of a DAG.



(b) The *DR-first* ordering rule.

Figure 2: A DAG-based SMR structured by three-round waves. (a) Each wave decides the blocks that will be committed (highlighted with different colors). (b) Nodes order blocks in wave w_j via the *DR-first* ordering rule, where the tree spawned by the anchor B_2^{i+2} and the LOP of each involved block jointly decide the final order of the blocks.

With a block ordering rule (see below), nodes will consistently order the historical blocks.

The block ordering rule. The DAG-based SMR relies on a block ordering rule to achieve zero communication overhead when ordering blocks. The block ordering rule can be any deterministic rule. For instance, many DAG-based SMR protocols (such as [18], [19], [21]) adopt an ordering rule that combines a Depth-first traversal and a Round-first reordering operation in their implementations[5]. (We therefore call it ***DR-first* ordering rule** in this work.) Figure 2b presents an example of ordering blocks via the *DR-first* ordering rule, which consists of two operations:

1) Depth-first traversal: Once an anchor B_2^{i+2} is ready to commit, each node constructs a tree spawned by B_2^{i+2} and all its historically uncommitted blocks, where (i) B_2^{i+2} is the root of the tree; (ii) the first-ordered block in B_2^{i+2}'s LOP is the root of B_2^{i+2}'s leftmost subtree, the second-ordered block of B_2^{i+2}'s LOP is the root of B_2^{i+2}'s second leftmost subtree, and so on. (Note that a block is only located at the tree when it is first traversed.) Nodes then collect the blocks of the tree with a pre-order depth-first traversal. For example, in Figure 2b, the temporary order after tree traversal is B_2^{i+2}, B_3^{i+1}, B_1^i, B_3^i, B_1^{i+1}, B_2^i, and B_2^{i+1}.

2) Round-first reordering: Since a DAG-based SMR creates blocks round-by-round, each node reorders the traversed blocks based on their round numbers, to partially capture the happen-before relationship. This ensures that if there is a connection (or path) from a block B_1 to B_2 (i.e., B_2 is created before B_1), then B_2 is eventually ordered before B_1. In Figure 2b, the eventual order after the round-first reordering is $B_1^i \prec B_3^i \prec B_2^i \prec B_3^{i+1} \prec B_1^{i+1} \prec B_2^{i+1} \prec B_2^{i+2}$, where $\prec$ represents an order-before relation between blocks.

5. An example can be found in https://github.com/facebookresearch/narwhal/blob/main/consensus/src/lib.rs.

2.2. Frontrunning and Existing Countermeasures

Frontrunning attacks. In the early-stage blockchain/SMR protocols, nodes jointly work to maintain two security properties (i.e., safety and liveness). However, neither of these properties captures order relationships among transactions while distinct orders can lead to different states of a ledger. The lack of rational order relationship makes *frontrunning* possible. Informally, in a frontrunning attack, an attacker aims to make its transaction tx_a ordered and executed before a victim transaction tx_v, regardless of the order between tx_a and tx_v when they are created and received by the network.

Frontrunning attacks have shown to be happening at a massive scale [2]–[12], [32]–[34]. Torres et al. [4] perform a large-scale analysis on three types of frontrunning, identifying almost 200K attacks with an accumulated profit of 18.41M USD. Zhou et al. [5] evaluate a variant of frontrunning attack (called sandwich attack), showing that a single adversarial trader can earn a daily revenue of over several thousand USD. Qin et al. [7] further demystify the dark forest of DeFi, showing the sandwich attack can yield over 174M USD over 32 months. The subsequent works show that frontrunning, involving actions outside blockchain (such as centralized exchanges [11] and rollups [12]), can also yield a large amount of profits.

Existing mitigations. To prevent frontrunning attacks, a line of consensus protocols [35]–[43] has been proposed. These solutions are motivated by a key observation: frontrunning strategies heavily rely on the transaction ordering rule. For instance, in a maximal extractable value-priority blockchain like Ethereum, attackers can either set a high transaction fee or work as the block builder (or proposer) to prioritize ordering their transactions. Therefore, to prevent frontrunning, the core idea of these mitigations is to replace the vulnerable ordering rule with a frontrunning-resilient ordering rule. For example, in Pompē [35], transactions are ordered based on their timestamps received by the majority of validators. Themis [37] orders transactions based on the local orders established by the majority of validators. Additionally, many other solutions [10], [32], [34] advocate to frontrun the frontrunning attackers and refund profits to victims.

Motivation. We observe that frontrunning attacks can be constructed strategically against the specific transaction ordering rules. Compared to the sing-chain-based blockchain, the DAG-based blockchain employs several new ordering features. First, to avoid duplicate transactions packed in multiple blocks, the DAG-based blockchain allows each transaction to be packed in the block by only one validator. Since validators cannot arbitrarily pick up transactions in their blocks, manipulating the order of transactions within a

block (i.e., intra-block frontrunning) becomes less possible. On the other hand, to order multiple blocks in the causal history of an anchor, the DAG-based blockchain introduces a new block ordering rule, which may expose the protocol to new frontrunning vulnerabilities. However, upon reviewing related studies, we find a notable absence of exploration and analysis of frontrunning attacks on DAG-based blockchains in the existing literature.

This work is conducted to fill this gap. Through in-depth analysis, we find even if intra-block frontrunning is less likely in DAG-based blockchains, attackers can perform a new frontrunning attack that frontrun victim transactions across blocks. We call it the inter-block frontrunning attack. Unlike intra-block frontrunning, inter-block frontrunning is more possible and powerful. First, inter-block frontrunning is independent of transaction assignments, meaning that an attacker can still frontrun the victim transaction even if the transaction is not assigned to it for ordering and commitment. Additionally, since inter-block frontrunning manipulates the global order of transactions, it can still work toward the victim transaction even if the transaction has already been intra-block frontrun. In the following sections, we will disclose the vulnerabilities of DAG-based blockchains to the inter-block frontrunning attack[6].

3. Problem Definition

This section formally defines a new inter-block frontrunning attack, followed by the threat and system models.

3.1. Inter-block Frontrunning Attack

As illustrated in Section 2.1, a DAG-based SMR relies on a deterministic block ordering rule to achieve efficient ordering for blocks. While guaranteeing all honest nodes maintain a consistent order for blocks, the ordering rule does not capture the rational order relationship that is resistant to order manipulations. We find that the lack of such property renders the existing DAG-based SMR protocols vulnerable to a new frontrunning attack, called *inter-block frontrunning attack*. On a high level, through manipulating the order of blocks, this attack can manipulate the order of transactions of these manipulated blocks, making the attacking transaction executed before the victim transactions. Formally, the inter-block frontrunning attack can be defined with a committing order relationship as follows:

Definition 3 (Block committing order)**.** *Let $\prec_{\mathbf{C}}$ denote a block committing order relation in a DAG-based SMR. Given two distinct blocks B_i and B_j, we define $B_i \prec_{\mathbf{C}} B_j$ if B_i is ordered before B_j after they are committed.*

Definition 4 (Inter-block Frontrunning)**.** *A block B_v is said to be frontrun by a block B_a from the attacker N_a if N_a creates B_a after witnessing B_v and eventually $B_a \prec_{\mathbf{C}} B_v$.*

6. For simplicity, we will use the term frontrunning attack to exclusively represent the inter-block frontrunning attack in the rest of this paper.

To elaborate further, the block committing order $\prec_{\mathbf{C}}$ defines an order relation only for blocks that are eventually committed. (Note that a block may not be committed if the block cannot be traversed by any anchor.) Moreover, we consider the frontrunning attack a purposeful order manipulation as previous works [2]–[5]. In other words, the inter-block frontrunning attack intentionally orders the attacking block B_a before the target block B_v containing specific content (e.g., involving significant cryptocurrency exchange), rather than arbitrarily ordering B_a before any other block. Therefore, B_a is created and propagated to the network later than B_v.

The goal of the attack. In our presented inter-block frontrunning attacking game, the goal of the attacker is to make its attacking block B_a eventually ordered and committed before the victim block B_v, i.e., achieving $B_a \prec_{\mathbf{C}} B_v$. As we will present in Section 4, the attacker can adopt some attacking strategies to improve the probability of achieving the attacking goal. Before delving into these attacking strategies, we define the threat and system models for a DAG-based SMR for illustration purposes.

3.2. Threat and System Models

This paper considers two types of adversaries: *Byzantine* and *frontrunning attacker*. Byzantine nodes are defined as adversaries that can behave arbitrarily to violate the security properties of the DAG-based SMR (i.e., safety and liveness), but are computationally bounded and cannot break standard cryptographic constructions. Frontrunning attackers are defined as adversaries that try to manipulate the order between blocks, i.e., launch the inter-block frontrunning attack. On the contrary, a node is *honest* if it is neither a Byzantine node nor a frontrunning attacker.

We consider a DAG-based SMR consisting of n nodes (or called validators). Under a non-synchronous network, the Byzantine fault tolerance of the DAG-based SMR t satisfies $t < n/3$ [18]–[22]. Moreover, we assume there are f_l ($f_l \leq t < n/3$) Byzantine nodes and f_a ($f_a < n$) frontrunning attackers. We allow up to $n-1$ frontrunning attackers to exist in the system. This is rational in practice because (i) every node seeking to maximize profits could potentially engage in frontrunning attacks [3], [44]; (ii) frontrunning attackers do not compromise the safety and liveness, i.e., the DAG-based SMR can still work regardless of f_a. Note that $f_a + f_l < n$ because we aim to ensure the presence of at least one honest node (i.e., victim) whose blocks can be frontrun in the attacking game. For illustration purposes, we assume the f_a frontrunning attackers are attacking the same victim block B_v, and the attacking block B_a created by a designated frontrunning attacker N_a is used to illustrate the attacking process. Nevertheless, frontrunning attackers are not aware of each other and might compete with each other.

The honest nodes and frontrunning attackers collaborate to build an ever-growing DAG, similar to how nodes operate in a DAG-based SMR without frontrunning attackers. For the consensus layer, this paper focuses on the *DR-first* ordering rule (illustrated in Figure 2b) as it is widely employed by

open-source DAG-based SMR protocols, such as Tusk [18], Bullshark [19], Mysticeti [21], and Sui lutris [25]. Additionally, the local ordering preference (LOP) is assumed to be based on the descending order of digests of blocks, similar to the implementations of Tusk [18] and Bullshark [19], which are representative since many subsequent DAG-based SMR protocols [20], [22], [25] claim to be built on them.

4. Inter-block Frontrunning Attacking Game

The block ordering rule presents frontrunning attackers with new opportunities to frontrun transactions in DAG-based SMR. In this context, the designated frontrunning attacker N_a aims to make its attacking block B_a ordered before a victim block B_v from node N_v even though B_a is created after B_v. The attacking process can be modeled as a game between a victim and frontrunning attackers, where the frontrunning attackers are said to win the game if they successfully frontrun B_v with the B_a. In this section, we give an insight into the winning rules of the game (Section 4.1). Guided by these rules, we exploit three attacking strategies for the frontrunning attackers to increase the probability of winning: the *fissure attack* (Section 4.2), the *speculative attack* (Section 4.3), and the *sluggish attack* (Section 4.4).

4.1. Winning Rules in the Game

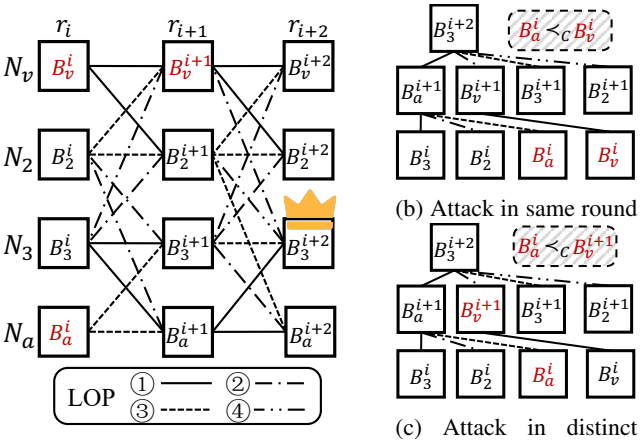
***DR-first* ordering rule revisited.** Referring back to the *DR-first* ordering rule, once a new anchor is selected, nodes traverse all historically uncommitted blocks of the anchor with a depth-first traversal and order the traversed blocks based on their round numbers. In this context, if a block is traversed first and associated with a smaller round number, then the block will eventually be ordered and committed first. This actually indicates two winning rules for the frontrunning attackers, where we consider both the attacking block B_a and the victim block B_v are ordered by the same anchor (i.e., they are traversed within the same tree)[7]:

Winning Rule 1 (WR1). *If both B_a and B_v belong to the same round, and B_a is to the left of B_v within the tree spawned by an anchor, then $B_a \prec_{\mathcal{C}} B_v$.*

Winning Rule 2 (WR2). *If B_a belongs to a round smaller than the round of B_v, then $B_a \prec_{\mathcal{C}} B_v$.*

Example: Figure 3 illustrates the winning rules for the frontrunning attackers who are frontrunning either the block B_v^i or B_v^{i+1} with the attacking block B_a^i. In Figure 3a, block B_3^{i+2} is selected as an anchor, and the *DR-first* ordering rule is employed to order the causal history of B_3^{i+2}. In the LOP of B_3^{i+2}, B_a^{i+1} is prioritized for traversal compared to B_v^{i+1}. As B_a^{i+1} connects to B_a^i but lacks a connection to B_v^i, traversal from B_a^{i+1} is feasible for B_a^i but not for B_v^i. Instead, B_v^i is traversed from B_v^{i+1}, indicating that B_v^i is to the right of B_a^i within the tree. According to the **WR1**, B_a^i is eventually ordered before B_v^i, as shown in Figure 3b. On the other hand, when B_v^{i+1} is the victim block that the attackers are frontrunning, B_a^i is associated with a smaller round than B_v^{i+1}. According to the **WR2**, B_a^i is eventually ordered before B_v^{i+1}, as shown in Figure 3c.

7. The scenarios where B_a and B_v are ordered by different anchors will be discussed in Appendix A, as these scenarios render the attack irregular.



(a) A temporary DAG structure

(b) Attack in same round

(c) Attack in distinct rounds

Figure 3: Illustration of the winning rules: (a) a DAG from round r_i to round r_{i+2}, containing an attacking block B_a^i and two victim blocks B_v^i and B_v^{i+1}; (b) block B_a^i is ordered before block B_v^i as B_a^i is to the left of B_v^i; (c) B_a^i is ordered before B_v^{i+1} as B_a^i has a smaller round number.

Attacks in a nutshell. According to the winning rules, if the frontrunning attackers want to launch a successful attack, their attacking block B_a must be either in the tree closer to the left or created in a smaller round (i.e., at a lower tree level) than the victim block B_v. To this end, the high-level idea of the frontrunning attack is to manipulate the formation of the tree where B_a and B_v are ordered, such that B_a is either to the left of B_v within the tree or at a lower level of the tree than B_v. In the following sections, we will present three attacking strategies that the frontrunning attackers can employ to manipulate the formation of a tree to increase their probability of winning the game.

4.2. Fissure Attack

According to the **WR1**, the frontrunning attack will be successful if B_a is situated in the tree further to the left than B_v. To construct a tree with blocks lying in specific positions, we observe that there exists a *connection priority*:

Observation 4.2.1 (Connection Priority). *Given two blocks B_a^i and B_v^i from the same round r_i that are ordered and committed by the same anchor, if B_a^i is connected (directly or indirectly) with more successor blocks than B_v^i, then B_a^i is more likely to be on the left of and traversed before B_v^i within the tree spawned by the anchor, and vice versa.*

The connection priority indicates the impact of subsequent connections on the ordering priority for blocks within the same round. To elaborate, a block B_1^i connected by more

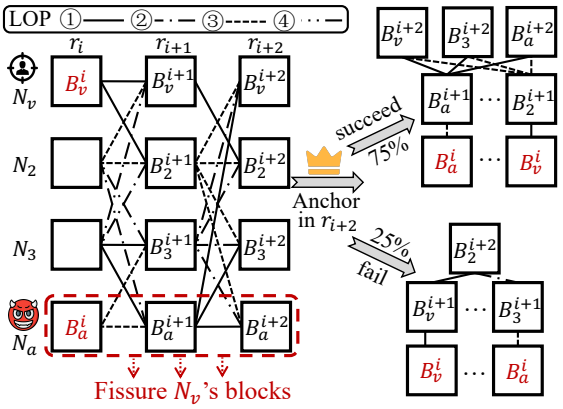


Figure 4: The fissure attack: the frontrunning attacker N_a creates a fissure between its blocks and the victim N_v's blocks by excluding N_v's blocks from its connections. With the fissure attack, there are more paths (i.e., 7 paths) to B_a^i than to B_v^i (i.e., 5 paths) from the anchor round r_{i+2}, leading that B_a^i is more likely ordered before B_v^i, i.e. 75% vs. 25%.

successor blocks (including direct connections from the next round and indirect connections via a path) means that more paths can reach B_a^i from the subsequent rounds, resulting in a higher probability that B_a^i is closer to the leftmost subtree. When employing the *DR-first* ordering rule, B_a^i is more likely to be traversed and therefore ordered before B_v^i.

Attacking Process. Based on the above observation, the frontrunning attackers can launch a *fissure attack* to make their block ordered before the victim block (as much as possible) by strategically selecting parent blocks for their created block. Specifically, the frontrunning attackers always *exclude the blocks created by the victim from the connections of their blocks*, such that the victim block has fewer connections than the attacking block.

Figure 4 illustrates an example of the fissure attack, where N_v is the victim and N_a is the frontrunning attacker. In this example, N_a is trying to use its attacking block B_a^i to frontrun the victim's block B_v^i. According to the connection priority (Observation 4.2.1), if B_a^i is connected with more successor blocks compared to B_v^i, then B_a^i is more likely to be traversed (and consequently ordered) before B_v^i, i.e., the frontrunning attack is more likely to succeed. To increase the successful probability of the attack, N_a creates a *fissure* between its blocks and N_v's blocks by excluding N_v's blocks from the connections of its blocks. With this fissure, N_v's blocks (including B_v^i) will never be directly traversed by N_a's blocks and therefore are less likely to be traversed before N_a's blocks. (Note that N_v's blocks can still be traversed by N_a's blocks indirectly, e.g., from the path $\langle B_a^{i+2}, B_2^{i+1}, B_v^i \rangle$.) To conclude, by constructing such a fissure, the number of paths reaching B_v^i decreases, thereby increasing the probability of traversing B_a^i before B_v^i. For instance, in Figure 4, the number of paths from blocks of round r_{i+2} to B_v^i is 5, less than 7 paths to B_a^i. Once selecting an anchor in round r_{i+2}, there are 75% probability leading to $B_a^i \prec_{\mathcal{C}} B_v^i$ (i.e., if one of B_v^{i+2}, B_3^{i+2}, and B_a^{i+2} is the anchor), which is higher than 25% probability that $B_v^i \prec_{\mathcal{C}} B_a^i$ (i.e., B_2^{i+2} is the anchor). In other words, the frontrunning attack is more likely to succeed.

4.3. Speculative Attack

In the formation of a tree, the local ordering preference (LOP) also affects the relative positions between blocks in the tree and therefore decides the order of blocks according to **WR1**. To elaborate, assume blocks B_a^i and B_v^i are connected by and traversed from a subsequent block B_3^{i+1}. If B_a^i is locally ordered before B_v^i in B_3^{i+1}'s LOP, then $B_a^i \prec_{\mathbf{C}} B_v^i$ when employing *DR-first* ordering rule. As claimed in Section 3.2, the LOP is implemented to sort blocks by descending order of their digests. Therefore, to construct a tree with blocks lying in specific positions, we observe that there exists a *digest priority*:

Observation 4.3.1 (Digest Priority). *Given two blocks B_a^i and B_v^i from the same round r_i that are ordered and committed by the same anchor, if B_a^i is with a larger digest than B_v^i, then B_a^i is more likely to be on the left of and traversed before B_v^i within the tree spawned by the anchor, and vice versa.*

Attacking Process. Given the above observation, the frontrunning attackers can manipulate the local order of blocks by manipulating the digests of their blocks, launching a so-called *speculative attack* to make the attacking block more likely to be ordered before the victim block. Briefly, when moving to a new round, the frontrunning attacker speculatively creates multiple new blocks and then picks the block with the largest digest to broadcast. As a larger digest is ordered first in the LOPs of the subsequent blocks, the new block picked by the frontrunning attacker speculatively is more likely to be ordered before the victim's block.

Figure 5 demonstrates an example of the speculative attack. Assume the frontrunning attacker N_a is on the same round r_i as the victim N_v when it receives the victim block B_v^i from N_v. To frontrun B_v^i, N_a first creates j frontrunning blocks $\{B_{a_1}^i, \cdots, B_{a_*}^i, \cdots, B_{a_j}^i\}$ belonging to the same round as B_v^i by some way, e.g., packing different transaction batches into blocks[8]. After that, N_a picks the newly created block with the maximum digest (assume it is $B_{a_*}^i$ in Figure 5) and sends it to the network. (Note that the other created blocks are abandoned and would not appear in the network to prevent equivocations.) With the speculative attack, it is more possible for the frontrunning attacker to create a new block with a larger digest than the digest of B_v^i. If this happens, then when moving to the next round r_{i+1}, nodes that receive both B_v^i and $B_{a_*}^i$ will order $B_{a_*}^i$ before B_v^i in their LOPs, leading to a successful frontrunning attack, as shown in the step (3) of Figure 5.

8. Note that in the implementations of [18], [19], the digest of a block is calculated with the round number, the name of the creator, the digests of parent blocks, and the metadata of transaction batches it piggybacks. Sampling distinct transaction batches is an easy and efficient (but not the only) way to obtain distinct blocks with different digests.

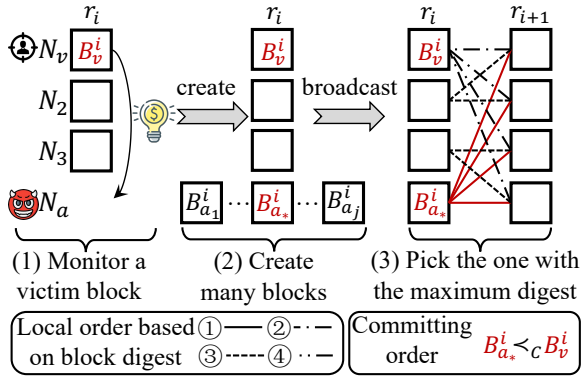


Figure 5: The speculative attack: the frontrunning attacker N_a speculatively creates multiple blocks in a round and only broadcasts the one with the maximum digest to the network.

4.4. Sluggish Attack

According to **WR2**, the frontrunning attack will be successful if B_a is created in a smaller round than B_v. This indicates a *round priority* in the *DR-first* ordering rule:

Observation 4.4.1 (Round Priority). *Given two blocks B_a^i and B_v^j from distinct rounds but are ordered and committed by the same anchor, if B_a^i is associated with a smaller round number than B_v^j (i.e., $r_i < r_j$), then B_a^i must be ordered before B_v^j, and vice versa.*

Attacking process. The frontrunning attacker can utilize this round priority to launch a so-called *sluggish attack*. Briefly, as blocks with smaller round numbers have ordering priority, the frontrunning attacker can intentionally delay its block creation and dissemination, thus slowing down the advancement of its round view. In this case, the designated frontrunning attacker N_a in a sluggish round can make its attacking block ordered before the victim block even though the victim block appeared in the network earlier. Figure 6 illustrates an example of the sluggish attack. Specifically, to frontrun the victim block B_v^{i+1}, N_a monitors the round number of the victim N_v and always keeps its round number lower than N_v's round number. This is feasible because: (i) when receiving a new block belonging to a round (e.g., r_{i+1}) from N_v, N_a can make sure that N_v must finish its round r_i; (ii) N_a then keeps its round number lower than r_i by delaying the creation of its block belonging to round r_i. Once N_a receives the target block B_v^{i+1} from N_v, it constructs and propagates its attacking block B_a^i. As N_a is under a smaller round than N_v, B_a^i will be associated with a smaller round number than B_v^{i+1}. Beneficial from the round priority, when both B_a^i and B_v^{i+1} are ordered and committed by the same anchor (i.e., the anchor B_2^{i+2} in Figure 6), B_a^i is ordered before B_v^{i+1}, i.e., the frontrunning attack succeeds.

5. Evaluation

In this section, we evaluate the effectiveness of the proposed frontrunning attack on two open-source DAG-based

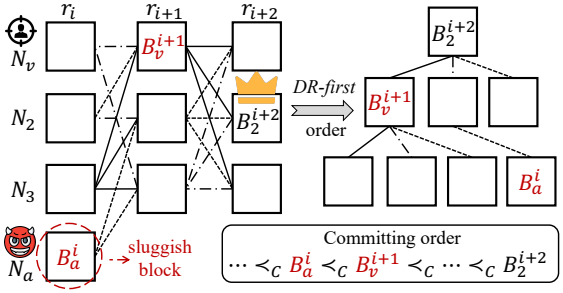


Figure 6: The sluggish attack: the frontrunning attacker N_a slows down to move to a new round so that it can create an attacking block B_a^i with a smaller round number than the victim block B_v^{i+1}.

SMR protocols, including Tusk [18] and Bullshark [19]. Through our evaluation, we identify some factors that can significantly influence the effectiveness of the attack. Notably, some of these factors are highly influential and can be readily manipulated by frontrunning attackers. This highlights the urgent need for the development of pertinent mechanisms to counteract this newly discovered attack.

5.1. Implementation and Evaluation Metric

We implement all three attacking strategies (discussed in Section 4) based on the codebase of each comparison DAG-based SMR protocol, i.e., Tusk[9] and Bullshark[10]. As many subsequent DAG-based SMR protocols (such as Shoal [20], Sailfish [22], and Sui lutris [25]) claim to be implemented based on Bullshark or Tusk, we believe our attack implementation and evaluation are representative. Tusk and Bullshark adopt a primary-worker architecture to disperse transactions and construct blocks. In the primary-worker architecture, there are f_w workers constructing transaction batches for each node. When creating blocks, nodes package the metadata (i.e., the hash) of the transaction batches received from workers into the blocks. Moreover, both Tusk and Bullshark adopt the *DR-first* ordering rule, and the LOP is based on the descending order of digests of blocks. The digest of a block is calculated with the round number, the name of the block creator, the digests of connected parent blocks, and the metadata of transaction batches it piggybacks. Additionally, similar to the evaluation settings of Tusk and Bullshark, the f_l Byzantine nodes are implemented as crashed nodes, i.e., they do not create blocks during the runtime of the protocol. We call these f_l Byzantine nodes *liveness attackers* in the rest of this section since they are working on breaking the liveness of DAG-based SMR. Furthermore, there are f_a frontrunning attackers in the protocol. Each frontrunning attacker in our implementation keeps monitoring blocks from the victims and can perform the following three attacking strategies:

9. https://github.com/asonnino/narwhal/

10. https://github.com/asonnino/narwhal/tree/bullshark. Note that this is an implementation of the partially synchronous version of Bullshark [31].

- **Fissure attack**: Whenever creating a new block, the frontrunning attacker excludes blocks of the victim node that it is attacking from the connections of its new block.
- **Speculative attack**: Whenever a frontrunning attacker is ready to create a new block, it first calculates at most p_{max} digests of blocks with different samples of transaction batches, where p_{max} is an attacking parameter used in our evaluation for the speculative attack. Then, the frontrunning attacker constructs the block with the maximum calculated digest and sends it to other nodes.
- **Sluggish attack**: The frontrunning attacker multiplies the timeout of creating a new block by an attacking factor t_s. By doing this, the attacker delays the process of block creation.

To evaluate the effectiveness of the frontrunning attacks using various attacking strategies-specifically, how much these attacking strategies can enhance the success rate of making the attacking block ordered before the victim block-we implement a baseline in the experiments:

- **Baseline**: In the baseline attack, a frontrunning attacker monitors the victim blocks and constructs attacking blocks without manipulating the order of blocks.

Due to the simplicity, the implementation of the above attacking strategies (including the baseline) is lightweight with about 700 lines of code in both Tusk and Bullshark.

Evaluation metric. In the following evaluation, we use the *attack success rate (ASR)* to evaluate the effectiveness of the frontrunning attack, which is calculated by the probability that frontrunning attackers eventually win the attacking game, i.e., they successfully frontrun the victim block.

5.2. Experimental Setup

Our evaluation is conducted to analyze the frontrunning attack happening across blocks. To this end, we need to collect the block data, including their creators, connected blocks, and committed positions[11]. However, we cannot derive the historically real-world block data from the deployed DAG-based blockchain (e.g., Sui) since their blockchain explorers (e.g., suiscan [45]) do not archive the block data. Collecting the block data by becoming a validator (with modified code) of the real-world DAG-based blockchain is economically unrealistic. For instance, a validator candidate must accrue at least 30M SUI of state (approximately $45M at the time of writing) before it can request to join in the validator set of Sui blockchain[12].

Since we cannot collect real-world block data for replay, we evaluate the proposed frontrunning attacks using real-time block data generated locally during our experiments. Specifically, we designate a victim node for each test to generate victim blocks randomly. All frontrunning attackers attack the same victim blocks by constructing corresponding attacking blocks upon monitoring the victim blocks. In our implementation, all blocks are associated with a new label *block height* to indicate the committed order of blocks. All block information, including the creators and block heights, is recorded in logs for the analysis of frontrunning attacks.

We conducted month-long experiments to comprehensively evaluate the effectiveness of the attack with different attacking strategies. These experiments were executed in both a single server (Section 5.3 - 5.5) and a geo-distributed AWS environment (for large-scale network evaluations, see Section 5.6). Specifically, the server runs Ubuntu 22.04 and is equipped with 48 CPU cores, 128GB of RAM, and 10TB SSD. Each measurement running in the server contains at least 500 times of attacks until we get a steady attack success rate. In the AWS setting, we deployed and ran our experiments in t3x.large EC2 instances, each of which has 4 CPU cores, 16GB of RAM, and a 5 Gbps bandwidth. Each measurement running in the AWS environment contains 100 times of attacks for monetary savings, yet we are able to show the incredible effectiveness of the attack. The two attacking parameters are respectively set by $p_{max} = 50$ and $t_s = 2$ as we observed from our evaluation that there are less than 50 digests the attacker can create in each round, and $t_s = 2$ is enough for the attacker to be processing in slower rounds than honest nodes.

5.3. Attack Effectiveness under Multiple Frontrunning Attackers

We first evaluate the effectiveness of different attacking strategies under varying ratios of frontrunning attackers f_a/n. Specifically, we set the ratio of frontrunning attackers $f_a/n = 0.1, 0.3, 0.5, 0.7, 0.9$. We conduct two groups of experiments with 10 nodes and 30 nodes respectively (i.e., $n = 10$ and $n = 30$). Figure 7 and 8 respectively show the attack success rates on Bullshark and Tusk. For each figure, the left two sub-figures represent results with $n = 10$ under varying numbers of liveness attackers f_l, and the right two sub-figures represent results with $n = 30$ under varying f_l. Note that in the figures, we start the y-axis at 30% (or 20%) to better illustrate the changes and differences.

Effectiveness of the fissure attack. From the experiment results, we observe that given n, f_l, and f_w, the fissure attack is more effective with increasing f_a. Incredibly, compared to the baseline (with around 48% ASR), the fissure attack can enhance the ASR by approximately 4% to 47% under varying ratios of frontrunning attackers, achieving up to 94.81% ASR on Bullshark. For Tusk, the fissure attack can enhance the ASR by up to 45% (compared to about 42% ASR of the baseline), achieving up to 87.31% ASR.

Moreover, we observe some exceptional cases. In the cases where only the victim node exists in the system (e.g., $n = 10, f_l = 0, f_a = 9$ or $n = 10, f_l = 2, f_a = 7$), the attackers are unable to frontrun the victim blocks. This is primarily because any anchors cannot traverse the victim block and, consequently, the victim block will not be committed. To elaborate, when $f_l + f_a = n - 1$, there are no honest nodes, except for the victim itself, to connect blocks

11. The committed position is same as the term *block height* used in Ethereum, which can be used to indicate the order among blocks.

12. https://docs.sui.io/guides/operator/validator-config

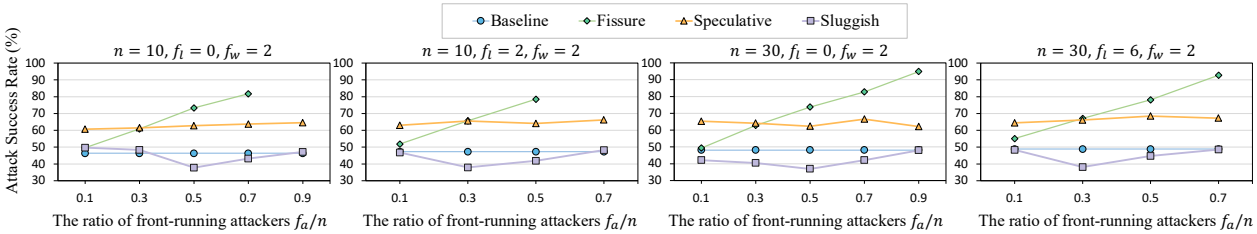


Figure 7: The attack effectiveness on Bullshark under varying numbers of frontrunning attackers f_a.

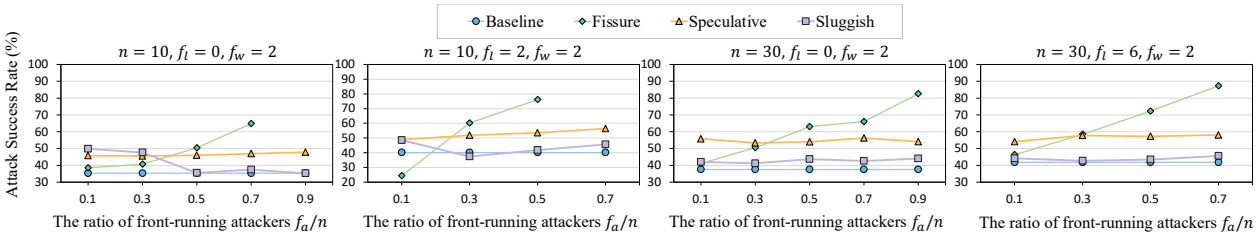


Figure 8: The attack effectiveness on Tusk under varying numbers of frontrunning attackers f_a.

from the victim. The victim block is therefore only traversed once a victim's block is selected as an anchor. However, the anchor belonging to the victim is not connected by enough subsequent blocks and is considered invalid. (Note that DAG-based SMR protocols consider a selected anchor valid only if it was connected by at least $t+1$ blocks, where $t = \lfloor \frac{n}{3} \rfloor$ is the Byzantine fault tolerance of the protocol.) Therefore, the victim block would never be ordered and committed. Furthermore, we observe the fissure attack even degrades the ASR on Tusk when $n = 10, f_l = 2, f_a = 1$ as shown in Figure 8. We analyze one hypothesis explaining this degradation case in Appendix C. Briefly, the attackers omitting the victim's blocks might require more time to move to new rounds and thereby make their blocks connected by fewer honest nodes than the victim block.

Effectiveness of the speculative attack. For both Bullshark and Tusk protocols, the speculative attack is affected slightly by the ratio of frontrunning attackers. By modeling the speculative attack in Appendix B.3, we show that the speculative attack is independent of the number of frontrunning attackers but is very relevant to the number of workers f_w (which will be also illustrated in Section 5.5). Nonetheless, compared to the baseline with only two workers (i.e., $f_w = 2$), the speculative attack can still enhance the ASR by about 18% on Bullshark and by about 15% ASR on Tusk.

Effectiveness of the sluggish attack. The sluggish attack exhibits fluctuations in its effectiveness based on the ratio of frontrunning attackers. Specifically, the sluggish attack can enhance the ASR when there are either very few or very many frontrunning attackers, but it may degrade the ASR otherwise For instance, under $n = 10$ and $f_l = 0$ (i.e., the leftmost sub-figure in both Figure 7 and 8), the sluggish attack can increase the ASR by 15% with $f_a = 1$ on Tusk but decrease the ASR by 8% with $f_a = 5$ on Bullshark. This is primarily because the sluggish attack is more likely to experience an *ordering delay* when many frontrunning attackers are slowing down the creation of blocks. When experiencing an ordering delay, the attacking block is ordered by a later anchor than the anchor that orders the victim block and therefore is ordered after the victim block (see Appendix A for more details).

5.4. Attack Effectiveness under Crash Faults

We then evaluate the attack effectiveness under the liveness attackers f_l. In our experiments, the liveness attackers do not create any blocks and perform as crash nodes. In this case, if the protocol decides an anchor belonging to a liveness attacker, then the selected anchor is invalid (in fact, it does not appear in the local DAG of any node), and the blocks of the corresponding wave will be ordered by a later valid anchor. To explore the impact of liveness attackers on the attack effectiveness, we set the ratio of liveness attackers $f_l/n = 0, 0.1, 0.2, 0.3$. Similarly, we conduct several groups of experiments with varying n and f_a.

Figure 9 and 10 respectively show the experiment results on Bullshark and Tusk. From these results, we observe that the ASRs increase overall for all attacking strategies with an increasing ratio of liveness attackers f_l/n. In particular, compared to the baseline with varying ratios of liveness attackers f_l/n, the fissure attack can enhance the ASR by approximately 14% to 18% under $f_a/n = 0.3$ and by approximately 26% to 32% under $f_a/n = 0.5$. For the comparison between the baseline and the speculative attack, the ASRs have been increased by about 15% to 20% with varying f_l/n on Bullshark and have been increased by about 11% to 17% with varying f_l/n on Tusk. The sluggish attack achieves lower ASR compared to the baseline with multiple frontrunning attackers due to the ordering delay (as explained in Section 5.3). However, the difference in the ASR between the sluggish attack and the baseline becomes smaller as more liveness attackers exist in the system.

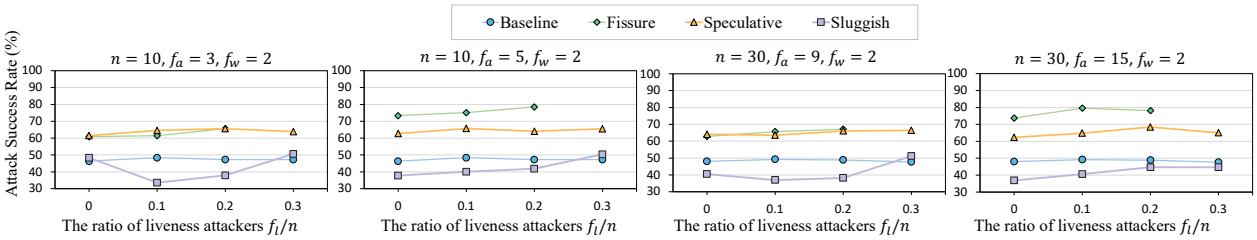


Figure 9: The attack effectiveness on Bullshark under varying ratios of liveness attackers f_l/n.

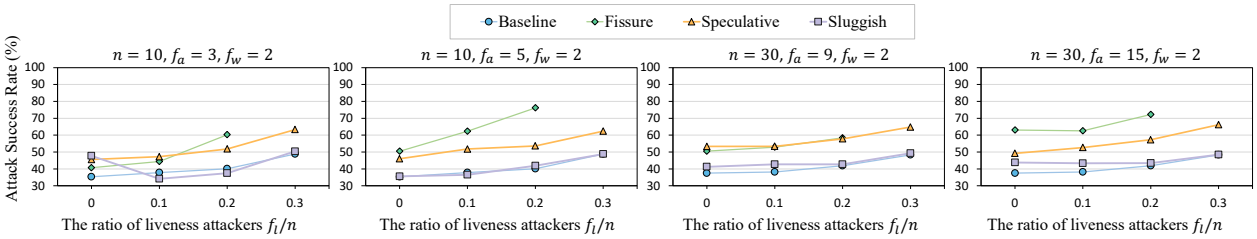


Figure 10: The attack effectiveness on Tusk under varying ratios of liveness attackers f_l/n.

This is because the victim block will be more likely to experience an ordering delay with more liveness attackers in the system. (Recall that anchors belonging to liveness attackers are invalid since they are not received and voted by other nodes.) As a consequence, both the victim block and the attacking block are still ordered and committed by the same anchor. The above experiment results prove the existence of the ordering delay and its impact on the effectiveness of the frontrunning attack. To conclude, the ratio of liveness attackers can affect the effectiveness of the attack by affecting the formation of ordering delays.

Moreover, we observe that the enhancement of ASRs on Bullshark is more modest compared to the more pronounced increase observed on Tusk. This is because Bullshark adopts a stronger round-moving rule in its partially synchronous version [31]. Specifically, different from Tusk where nodes move to the next round right after receiving $n - t$ blocks, nodes of Bullshark set a timer to wait for an anchor. In this case, the attacking block in Bullshark has more time to be received by other nodes and therefore is less likely to experience the ordering delay. According to the conclusion above, the attack effectiveness in Bullshark is affected slightly by liveness attackers.

5.5. Scale-out with Workers

We next evaluate the impact of workers on the attack effectiveness. In the DAG-based SMR protocol, workers are originally introduced to scale out the system by creating retrieval transaction batches for nodes. We find that workers in our proposed attacking game can affect the attack effectiveness as well. Intuitively, with more workers, the speculative attack can speculatively construct more blocks (with different transaction batches)[13] and therefore is more likely to create a block with a larger digest than that of the victim block. We prove this theoretical speculation in this section. Specifically, we conduct a group of experiments by setting the number of workers $f_w = 1, 2, 4, 8$ respectively. Figure 11 shows the effectiveness of various attacking strategies under varying numbers of workers connected by a node f_w.

From the experiment results, we observe that the speculative attack becomes more effective on both Bullshark and Tusk with an increasing number of workers connected by a node f_w, while the effectiveness of the fissure attack and the sluggish attack is independent of f_w. Specifically, the ASR of the speculative attack can reach up to 84.08% under $f_w = 8$ on Bullshark. Compared to the baseline (around 46% ASR), the speculative attack can enhance the ASR by 38%. Besides, for Tusk, the speculative attack can enhance the ASR by approximately 33% for the baseline under $f_w = 8$, achieving up to 73.02% ASR. These experiment results show the incredible effectiveness achieved by the speculative attack. Moreover, the enhancement of the attack effectiveness is positively related to the number of workers. As a consequence, while the DAG-based protocol creates more workers to help improve performance, such a scale-out architecture also makes the protocol more vulnerable to the frontrunning attack.

5.6. Attack Effectiveness under WAN Setting

We finally evaluate our attacks on the wide area network (WAN) setting, where we run nodes on AWS EC2 instances across 6 regions (3 in North America and 3 in Europe).

13. More precisely, the speculative attack is impacted by the number of new transaction batches per round, which is determined by not only f_w but also some node parameters, e.g., the delay of creating batches. We simplify our experiments here and give more analysis in Appendix B.1

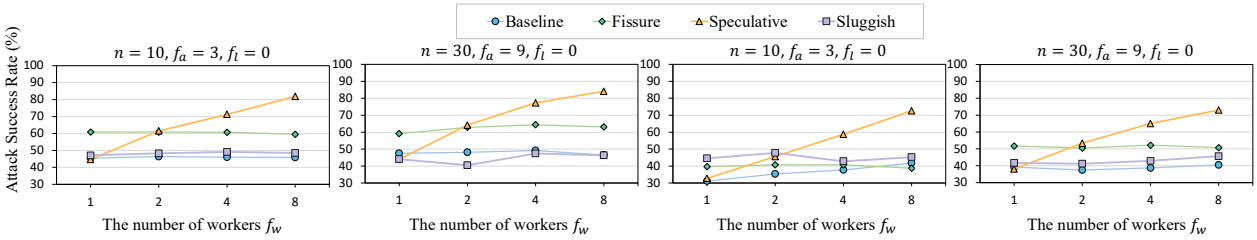


Figure 11: The attack effectiveness on Bullshark (left two) and Tusk (right two) under varying numbers of workers f_w.

From previous experiment results, we find different attacking strategies are affected by different factors. For example, the fissure attack is mainly affected by the ratio of frontrunning attackers f_a/n, while the speculative attack is mainly affected by the number of workers f_w. To better reflect attacking strategies deployed in practice, in the following experiments, we set half of the nodes frontrunning attackers (each of which is associated with 1 worker) for the fissure and sluggish attacks. For the speculative attack, we set $1/3$ nodes frontrunning attackers, each of which is associated with analogous 8 workers.

Attacks in various network scales. We first evaluate the scalability of our attacks, i.e., the attack effectiveness under various network scales. To this end, we set different numbers of nodes n =30, 52, 79, and 100 (same-level network scale as Sui). The experiment results are shown in Table 1, from which we find the speculative attack is the most effective and can achieve 92.86% ASR on Bullshark and 86.27% ASR on Tusk in a realistic network scale $n = 100$. Additionally, the ASRs of the fissure and speculative attacks enhance with the increasing network scale. One hypothesis explanation is that frontrunning attackers can benefit from the increasing delay of moving to new rounds as the network scales. To elaborate, for instance, since workers create transaction batches at a fixed speed, a speculative attacker can create more speculative blocks if the delay of moving to new rounds increases. With more speculative blocks, the speculative attacker is more likely to inter-block frontrun the victim block.

Attacks under crash faults. We then evaluate the impact of crash faults on the attack effectiveness under a realistic network, where we set $n = 30$. Table 2 concludes the experiment results. We can observe that the speculative attack is still the most effective. In particular, the speculative attack can achieve 87.5% ASR on Bullshark and 85.32% ASR on Tusk, which highlights the critical need to design countermeasures for this inter-block frontrunning attack. In the next section, we will discuss countermeasures.

TABLE 1: Attack success rate (%) with varying numbers of nodes n under the WAN setting.

	Bullshardk				Tusk			
n	30	52	79	100	30	52	79	100
Baseline	50.76	48.53	49.23	47.62	39.81	40.38	44.19	44.23
Fissure	73.23	79.07	80.26	78.72	60.95	66.67	62.79	67.31
Speculative	85.19	89.66	92.98	92.86	81.08	85.42	87.76	86.27
Sluggish	41.28	39.02	40.0	39.13	35.64	42.86	41.07	47.62

TABLE 2: Attack success rate (%) with varying ratios of crashed nodes (i.e., f_l/n) under the WAN setting.

	Bullshardk			Tusk		
f_l/n	0	0.1	0.2	0	0.1	0.2
Baseline	50.76	46.36	51.43	39.81	34.21	38.18
Fissure	73.23	72.93	81.97	60.95	66.97	66.67
Speculative	85.19	87.5	85.59	81.08	82.52	85.32
Sluggish	41.28	41.44	41.59	35.64	40.18	41.96

6. Mitigations

In this section, we will discuss some mitigations against the inter-block frontrunning attack on DAG-based SMR protocols, as well as their limitations. It is worth noting that the term *frontrunning* is not a new concept on the blockchain but the existing exploited frontrunning attacks focus on manipulating the order of transactions *within a block*. Essentially, the inter-block frontrunning attack is similar to the traditional (intra-block) frontrunning attack, where both aim to place the attacker's transactions before the victims' transactions. Therefore, motivated by existing countermeasures towards the intra-block frontrunning attack, we can mitigate the inter-block frontrunning attack from two directions: *order fairness* and *content agnostic*.

Order fairness. The first one is to introduce the order-fairness property to the block ordering rule as many order-fairness consensus protocols do [35]–[40]. The idea of preventing frontrunning attacks with order-fairness property is to define a reasonable ordering indicator based on the time when blocks are created so that the frontrunning block (which is created later than the victim block) cannot be ordered before the victim block. For instance, similar to Pompē [35], each node creates blocks piggybacked on the signed timestamps from other nodes, with which nodes then order blocks based on the middle timestamp of their piggybacked signed timestamps. The timestamp-based mitigation can guarantee the attacker cannot frontrun blocks as long as other honest nodes have a consistently received order for blocks; however, it will fail if honest nodes are not sufficiently synchronized or suffer from some network-level attacks (e.g., the adversary delays messages among honest nodes). Furthermore, we can also refer to the solutions from [36], [37], [40], where each node first locally orders blocks according to the receiption time, and then these local orders are integrated into a global order via a fair algorithm. However, these solutions require a longer time to order and

commit blocks because nodes must wait to receive enough local orders to determine a global order.

Content-agnostic ordering. Another direction to prevent frontrunning attacks is to integrate the content-agnostic ordering into the block ordering rule [46]–[48]. The high-level idea is to hide the content of transactions/blocks via some encryption algorithms and reveal the transactions/blocks after they are ordered. In this case, the frontrunning attacker might neither construct the attacking blocks nor manipulate the order among blocks since it cannot recognize the "target block" containing the transactions it wants to frontrun. For instance, Fino [47] integrates a secret sharing scheme into the DAG-based SMR protocol, where each user first encrypts the content of a transaction and secret shares its key to nodes, and eventually, nodes decrypt the transaction after it is ordered. However, the content-agnostic ordering solutions have several intrinsic limitations. First, they introduce extra computation and communication overhead for both users and nodes. Second, they cannot prevent content-agnostic frontrunning attacks [49], where the frontrunning attacker just wants to order its block before another block from its perceived victim node.

Non-deterministic ordering. The existing frontrunning mitigations can be somehow followed, but they all have limitations when applied to prevent our inter-block frontrunning attack in DAG-based SMR protocols, as discussed above. We find that there is another simple mitigation aiming to resist the inter-block frontrunning attack. Specifically, the frontrunning strategies proposed in Section 4 are severely relying on the block ordering rule. As the block ordering rule is deterministic (and public), the frontrunning attackers can *intentionally* construct a DAG that prioritizes ordering their blocks with our proposed attacking strategies. Therefore, to prevent such intentional behaviors, the DAG-based SMR protocol can adopt a *random* block ordering rule, which combines randomness generated in rounds to order all historically uncommitted blocks of each anchor. For instance, one can hash the randomness and the block digest, and order blocks based on the values of the hash results. Since the hash results are random (but note that they are identical for every node), there will be no ordering priorities that are originally constructed by the attacking strategies. Eventually, the attackers will not achieve a considerable success rate with the proposed attacking strategies. However, such a random block ordering rule inevitably ignores the happen-before relationship among blocks (i.e., a block B_1 should be ordered before another block B_2 if B_1 is connected by B_2 or there is a path from B_2 to B_1), which may lead to some application-layer problems, e.g., many transactions with data dependencies may experience execution failure and have to rollback. Additionally, such hashing operations will introduce extra computation overhead.

Transactions reordering. Moreover, nodes can globally reorder transactions after the *DR-first* ordering rule to eliminate the impact of inter-block frontrunning. For instance, the Sui blockchain currently reorders all transactions based on their gas fees after validators collect all transactions of blocks via the *DR-first* ordering rule. This reordering operation can make our proposed attacking strategies ineffective as they only manipulate the order of blocks generated by the *DR-first* ordering rule. However, it may make the protocol more vulnerable to frontrunning attacks using the existing attacking strategies, e.g., by setting higher transaction fees. Specifically, after monitoring a victim transaction TX_v from a validator's block, the frontrunning attacker can construct a block including the attacking transaction Tx_a. To make frontrunning succeed, the frontrunning attacker only needs to set the gas fee of TX_a higher than that of TX_v without crafting the formation of DAG as our proposed attacks do. Note that such a gas-based attacking strategy in DAG-based blockchains is even more effective compared to that employed in non-DAG blockchains. This is because, unlike the existing frontrunning attacks that could fail if Tx_a (even with a high gas fee) is not included in the same block as TX_v, the frontrunning attack in the DAG-based blockchain allows Tx_a and Tx_v to be included in distinct blocks.

7. Conclusion and Discussion

In this paper, we present a new frontrunning attack on DAG-based blockchains. We further discover three attacking strategies that the attackers can employ to increase the attack success rate. Our extensive experiments show remarkable attack effectiveness, highlighting the urgency of designing an effective countermeasure against the proposed attack.

Consensus instability and performance loss. Inter-block frontrunning attacks may further cause impacts on the consensus stability and performance of DAG-based blockchains. Specifically, since the block ordering rule is deterministic, a frontrunning attacker knows whether its attacking block will benefit from the causal order of a received leader block B_L^i. If B_L^i implies that the attacker's block will be ordered before the victim block, the attacker includes B_L^i in its newly created block in the next round r_{i+1} (i.e., votes for the leader block). Otherwise, the attacker excludes B_L^i. This *vote-for-favor* behavior can cause a leader block to fail to receive sufficient votes for commitment, ultimately preventing the leader block and its causal history from being committed in its wave. Additionally, given that the inter-block frontrunning necessitates the attacking block associating a round equal to and smaller than that of the victim block, inter-block frontrunning attackers may deliberately slow down their block creation until they detect a victim block. This strategy compromises both the transaction throughput and the confirmation latency of DAG-based blockchains.

Attack contentions. In many frontrunning scenarios, frontrunning attackers compete to attack the same victim transactions, leading to the failure of many attacking transactions to be committed. For instance, in Ethereum, multiple builders craft their block that orders their frontrunning transaction before the victim transaction. As Ethereum only allows one block to be committed each time (i.e., in each consensus instance), only one of the competing attackers can win in the frontrunning game while the others fail to commit their frontrunning transactions. Unfortunately, such attack contentions have little impact on inter-block frontrunning

attackers in DAG-based blockchains. Specifically, DAG-based blockchains allow multiple blocks to be proposed and committed in each consensus instance. Even though attackers compete to inter-block frontrun the same victim blocks with their own attacking blocks, their attacking strategies are independent and don't interfere with each other for block commitment (e.g., the speculative attack only crafts the attacker's own block). As a result, all attacking transactions can be executed and committed before the victim transaction. In our experiments, attackers frontrun the same victim blocks independently without communicating with each other, which indicated the attack effectiveness under attack contentions.

Specific frontrunning strategies. The proposed attacking strategies in § 4 are specifically designed to exploit the *DR-first* ordering rule, which has been employed in many open-source DAG-based blockchains, such as Tusk, Bullshark, and Mysticeti. If a DAG-based blockchain deploys a different block ordering rule, these strategies will become ineffective. However, frontrunning opportunities may still arise if no protections are employed, particularly in block ordering rules that are *pre-defined and deterministic*, where adversaries can intentionally manipulate the block order to their advantage. This work aims to reveal the frontrunning vulnerabilities of the ordering rule in DAG-based blockchains. Due to the limited availability of open-source DAG blockchains and the vagueness of the block ordering rules in the literature, we have not identified other block ordering rules currently in use. This leaves a big space for future exploration of various inter-block frontrunning strategies in DAG-based blockchains.

Acknowledgment

We thank Alberto Sonnino for his insights on the MEV and frontrunning issues on DAG-based blockchains. This work is supported in part by the National Science Foundation (NSF) under grant CNS1846316 and a research gift by Supra Labs.

References

[1] DeFiLlama. Defilama - defi overview. https://defillama.com/. Accessed: 2024.

[2] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. In *FC*, pages 170–189. Springer, 2019.

[3] 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 *IEEE S&P*, pages 910–927, 2020.

[4] Christof Ferreira Torres, Ramiro Camino, et al. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *USENIX Security*, pages 1343–1359, 2021.

[5] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *IEEE S&P*, pages 428–445, 2021.

[6] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the just-in-time discovery of profit-generating transactions in defi protocols. In *IEEE S&P*, pages 919–936, 2021.

[7] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *IEEE S&P*, pages 198–214, 2022.

[8] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. Sok: Decentralized finance (defi) attacks. *IEEE S&P*, pages 2444–2461, 2023.

[9] Robert McLaughlin, Christopher Kruegel, and Giovanni Vigna. A large scale study of the ethereum arbitrage ecosystem. In *USENIX Security*, pages 3295–3312, 2023.

[10] Kaihua Qin, Stefanos Chaliasos, Liyi Zhou, Benjamin Livshits, Dawn Song, and Arthur Gervais. The blockchain imitation game. In *USENIX Security*, pages 3961–3978, 2023.

[11] Lioba Heimbach, Vabuk Pahari, and Eric Schertenleib. Non-atomic arbitrage in decentralized finance. In *IEEE S&P*, pages 224–224, 2024.

[12] Christof Ferreira Torres, Albin Mamuti, Ben Weintraub, Cristina Nita-Rotaru, and Shweta Shinde. Rolling in the shadows: Analyzing the extraction of MEV across layer-2 rollups. *arXiv preprint arXiv:2405.00138*, 2024.

[13] libMEV. MEV - dashboard. https://libmev.com/. Accessed: 2024.

[14] Zihao Li, Jianfeng Li, Zheyuan He, Xiapu Luo, Ting Wang, Xiaoze Ni, Wenwu Yang, Xi Chen, and Ting Chen. Demystifying defi MEV activities in flashbots bundle. In *ACM CCS*, pages 165–179, 2023.

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

[16] Lioba Heimbach, Lucianna Kiffer, Christof Ferreira Torres, and Roger Wattenhofer. Ethereum's proposer-builder separation: Promises and realities. In *ACM IMC*, pages 406–420, 2023.

[17] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *ACM PODC*, pages 165–175, 2021.

[18] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a DAG-based mempool and efficient bft consensus. In *ACM EuroSys*, pages 34–50, 2022.

[19] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG bft protocols made practical. In *ACM CCS*, pages 2705–2718, 2022.

[20] Alexander Spiegelman, Balaji Aurn, Rati Gelashvili, and Zekun Li. Shoal: Improving DAG-bft latency and robustness. *arXiv preprint arXiv:2306.03058*, 2023.

[21] Kushal Babel, Andrey Chursin, George Danezis, Lefteris Kokoris-Kogias, and Alberto Sonnino. Mysticeti: Reaching the limits of latency with uncertified DAGs. *arXiv preprint arXiv:2310.14821*, 2023.

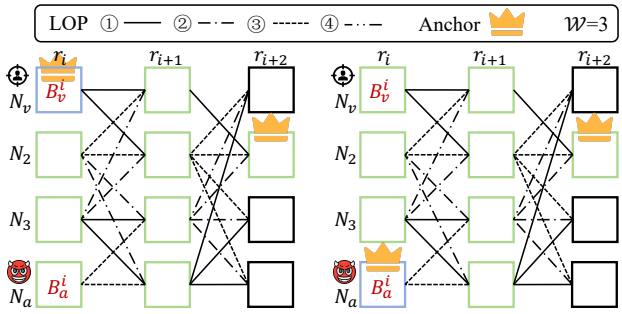
[22] Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving latency of DAG-based bft. In *IEEE S&P*, 2025.

[23] Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. Bbca-chain: One-message, low latency bft consensus on a DAG. *arXiv preprint arXiv:2310.06335*, 2023.

[24] Idit Keidar, Oded Naor, Ouri Poupko, and Ehud Shapiro. Cordial miners: Fast and efficient consensus for every eventuality. In *DISC*, 2023.

[25] Same Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, et al. Sui lutris: A blockchain combining broadcast and consensus. In *ACM CCS*, 2024.

[26] Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput DAG bft can be fast! *arXiv preprint arXiv:2405.20488*, 2024.



(a) The victim block B_v^i is the anchor (b) The attacking block B_a^i is the anchor

Figure 12: The conditions where an involved block is designated as an anchor (different colored blocks represent distinct waves): (a) the attack is failed if the victim block B_v^i is designated as an anchor; (b) the attack is successful if the attacking block B_a^i is designated as an anchor.

[27] The Sui Team. The sui blockchain. https://sui.io/. Accessed: 2024.

[28] Suiscan. Sui blockchain defi. https://suiscan.xyz/mainnet/directory?cat=DEFI. Accessed: 2024.

[29] The Aptos Team. Aptos networks. https://aptosfoundation.org/. Accessed: 2024.

[30] The Celo Team. Celo networks. https://celo.org/. Accessed: 2024.

[31] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: The partially synchronous version. *arXiv preprint arXiv:2209.05633*, 2022.

[32] Zhuo Zhang, Zhiqiang Lin, Marcelo Morales, Xiangyu Zhang, and Kaiyuan Zhang. Your exploit is mine: instantly synthesizing counter-attack smart contract. In *USENIX Security*, pages 1757–1774, 2023.

[33] Wuqi Zhang, Zhuo Zhang, Qingkai Shi, Lu Liu, Lili Wei, Yepang Liu, Xiangyu Zhang, and Shing-Chi Cheung. Nyx: Detecting exploitable front-running vulnerabilities in smart contracts. In *IEEE S&P*, pages 146–146, 2024.

[34] Chaofan Shou, Yuanyu Ke, Yupeng Yang, Qi Su, Or Dadosh, Assaf Eli, David Benchimol, Doudou Lu, Daniel Tong, Dex Chen, et al. Backrunner: Mitigating smart contract attacks in the real world. *arXiv preprint arXiv:2409.06213*, 2024.

[35] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *USENIX OSDI*, pages 633–649, 2020.

[36] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *CRYPTO*, pages 451–480. Springer, 2020.

[37] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. In *ACM CCS*, pages 475–489, 2023.

[38] Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *ACM AFT*, pages 25–36, 2020.

[39] Christian Cachin, Jovana Mićić, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness. In *FC*, pages 316–333. Springer, 2022.

[40] Ke Mu, Bo Yin, Alia Asheralieva, and Xuetao Wei. Separation is good: A faster order-fairness byzantine consensus. In *NDSS*, 2024.

[41] Wuhui Chen, Yikai Feng, Jianting Zhang, Zhongteng Cai, Hong-Ning Dai, and Zibin Zheng. Auncel: Fair byzantine consensus protocol with high performance. In *IEEE INFOCOM*, pages 1–10, 2024.

[42] Heena Nagda, Shubhendra Pal Singhal, Mohammad Javad Amiri, and Boon Thau Loo. Rashnu: Data-dependent order-fairness. *VLDB Endowment*, 17(9):2335–2348, 2024.

[43] Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. In *ACM ASIA-PKCW*, pages 3–14, 2022.

[44] Sarisht Wadhwa, Luca Zanolini, Francesco D'Amato, Aditya Asgaonkar, Chengrui Fang, Fan Zhang, and Kartik Nayak. Data independent order policy enforcement: Limitations and solutions. In *ACM CCS*, 2024.

[45] Suiscan. Sui blockchain explorer. https://suiscan.xyz/mainnet/home. Accessed: 2024.

[46] Peyman Momeni, Sergey Gorbunov, and Bohan Zhang. Fairblock: Preventing blockchain front-running with minimal overheads. In *SecureComm*, pages 250–271. Springer, 2022.

[47] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (MEV) protection on a DAG. *arXiv preprint arXiv:2208.00940*, 2022.

[48] Haoqian Zhang, Louis-Henri Merino, Vero Estrada-Galinanes, and Bryan Ford. Flash freezing flash boys: Countering blockchain front-running. In *IEEE ICDCS Workshop*, pages 90–95, 2022.

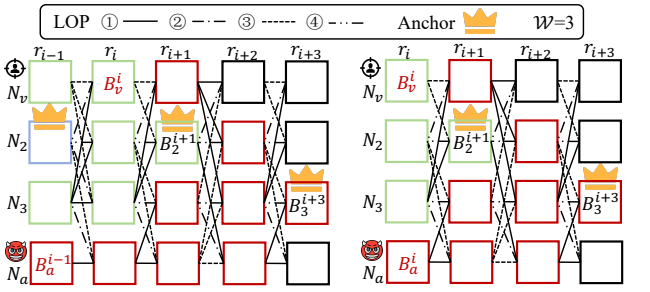
[49] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: MEV countermeasures: Theory and practice. *arXiv preprint arXiv:2212.05111*, 2022.

Appendix A. Attacks across Anchors

The attacking strategies presented in Section 4 only consider the condition where both the victim block B_v and the attacking block B_a are ordered and committed by *the same anchor*. However, the construction of the DAG is subject to asynchronous block dissemination, leading to arbitrary connections between blocks. In some cases, B_v and B_a may be ordered and committed by different anchors no matter if they belong to the same round (for fissure and speculative attacks) or different rounds (for sluggish attack), leading that the *attack involves across anchors*. In the following, we will discuss distinct scenarios where the frontrunning attack is across anchors and analyze their impacts on the successfully attacking probability.

(1) B_v or B_a is the anchor: Once B_v and B_a belong to the same round and one of them is selected as an anchor, the frontrunning attack is across anchors. In this case, the block selected as the anchor will be prioritized for ordering and commitment, preceding the other block, which must await ordering by the subsequent anchor. Figure 12 demonstrates two distinct scenarios where B_v and B_a are designated as the anchor respectively. In Figure 12a, the victim block B_v^i is designated as the anchor in round r_i, and if both B_v^i and B_a^i are eventually committed, then B_v^i is ordered before B_a^i, i.e., the frontrunning attack fails. In contrast, if the attacking block B_a^i is designated as the anchor, and both B_v^i and B_a^i are eventually committed, then B_a^i is ordered before B_v^i, leading to a successful attack (as shown in Figure 12b).

(2) Ordering delay: The precondition for a block to be ordered and committed by an anchor is the existence of a path between the block and the anchor, i.e., the block can be traversed from the anchor. As a consequence, if a block is not traversable for the anchor within the same wave, it will undergo a delay in being ordered and committed until subsequent anchors undertake the ordering task. This can result in an attack across anchors, where B_v and B_a are



(a) The attacking block B_a^{i-1} is ordered after the victim block B_v^i due to ordering delay

(b) The attacking block B_a^i is ordered after the victim block B_v^i due to ordering delay

Figure 13: The conditions where an involved block experiences a delay in being ordered (different colored blocks represent distinct waves): (a) the attacking block B_a^{i-1} is not traversable for the nearest anchor B_2^{i+1} and therefore is delayed for ordering; (b) the attacking block B_a^i is not traversable for the nearest anchor B_2^{i+1} and therefore is delayed for ordering.

ordered and committed by distinct anchors designated in different waves. Figure 13 presents two scenarios where B_v and B_a are ordered and committed by different anchors.

In Figure 13a, the victim block B_v^i belongs to round r_i while the attacking block B_a^{i-1} is created in round r_{i-1} (assume after the sluggish attack). However, since B_a^{i-1} experiences an ordering delay, B_a^{i-1} is eventually ordered later than B_v^i even though B_a^{i-1} is associated with a smaller round number, i.e., the frontrunning attack with the sluggish attacking strategy is failed. To elaborate, since block B_2^{i+1} is selected as an anchor in round r_{i+1}, and B_v^i is traversable for B_2^{i+1} but B_a^{i-1} is not, B_v^i can be ordered and committed by B_2^{i+1} while B_a^i will be ordered and committed by a subsequent anchor B_3^{i+3} in round r_{i+3}. Similarly, in Figure 13b, where both the victim block B_v^i and the attacking block B_a^i belong to round r_i, the attacking block B_a^i is experiencing an ordering delay while the victim B_v^i is not, and therefore B_a^i is ordered after B_v^i, i.e., the attack fails.

Appendix B. Attacking Strategies Modeling and Analysis

To better understand the effectiveness of the inter-block frontrunning attack under different system parameters in practice and design effective countermeasures, we formally model and generalize the attacking strategies proposed in Section 4. We emphasize that our current model ignores the network-layer attacks (Section 7) because we only focus on the impact of our proposed attacking strategies on the attack effectiveness; otherwise, the network-layer attacks will make the formation of the DAG unpredictable, and we cannot evaluate the attack effectiveness based on the unpredictable DAG. Note that due to space constraints, we omit the equation simplification process and include them in the full version of the paper.

B.1. System Model

We model the inter-block frontrunning attack as a game among n nodes $\mathcal{N} = \{N_1, \cdots, N_n\}$ with f_a ($f_a < n$) collusive frontrunning attackers $\mathcal{A} = \{N_{a_1}, \cdots, N_{a_{f_a}}\}$ and f_l ($f_l \leq t$) liveness attackers $\mathcal{L} = \{N_{l_1}, \cdots, N_{l_{f_l}}\}$. Besides, each node is associated with f_w workers, and the DAG interprets every $\mathcal{W}$ consecutive rounds as a wave. Furthermore, as we will show in the following section, the number of rounds required to order and commit B_v and B_a can impact the attacking probability. Therefore, we define ∇_r as the number of rounds between B_v and the nearest anchor round after which both B_v and B_a are committed. For instance, $\nabla_r = 2$ in Figure 12a while $\nabla_r = 3$ in Figure 13a.

In this game, there is a victim block B_v created by a victim node N_v and an attacking block B_a created by a designated frontrunning attacker N_a. It is said that $\mathcal{A}$ win the game if B_a successfully frontruns B_v. This game has the following properties:

(1) Prop1: *Random anchors.* The anchors are selected randomly, in which every node's block in the anchor round shares the same probability of being selected as an anchor.

(2) Prop2: *Random digest.* The hash function used to calculate the block digests can produce uniformly distributed outputs (i.e., ensuring uniformity). In this case, for any block digest, there is a 50% probability that a newly created block digest will be greater than it.

(3) Prop3: *Predictable transaction batches.* In the data dissemination layer, each worker generates one transaction batch in each round. The number of transaction batches that can be packed into a new block (denoted n_{bat}) is predictable and equal to the number of workers f_w.

Even though in practice, n_{bat} can also be affected by several parameters (such as the delay of creating a block, the delay of creating a transaction batch, and the transaction rate), we assume $n_{bat} = f_w$ to simplify our model. This assumption is consistent with our observations in the evaluation, and we believe that this simplification does not affect the evaluation of the attack effectiveness.

(4) Prop4: *Time-relevant ordering delay.* A block is more likely to experience an ordering delay if it is created and dispersed later than another block. This is because nodes in the DAG-based blockchains will move to the next round once they receive $n - t$ blocks from the current round. Consequently, a slow block will be excluded from the connections of their newly created blocks and will experience an ordering delay.

Attack across anchors. In the following sections, we will model the inter-block frontrunning attack under distinct attacking strategies while considering the conditions where the attack is across anchors (Appendix A). For illustration purposes, we define the following *mutually exclusive conditions* for the victim block B_v and the attacking block B_a:

$\Phi_1 : B_v$ is an anchor; $\Phi_2 : B_a$ is an anchor;

$\Phi_3 : B_v$ is ordered after B_a due to the ordering delay;

$\Phi_4 : B_a$ is ordered after B_v due to the ordering delay;

Let $Pr(\Phi_1)$, $Pr(\Phi_2)$, $Pr(\Phi_3)$, and $Pr(\Phi_4)$ denote the probabilities of Φ_1, Φ_2, Φ_3, and Φ_4 respectively. Since each

wave will select an anchor in the anchor round, we have $Pr(\Phi_1) = Pr(\Phi_2) = \frac{1}{\mathcal{W}} \cdot \frac{1}{n} = \frac{1}{\mathcal{W}n}$ according to **Prop1**. On the other hand, Φ_3 and Φ_4 indicate that B_v and B_a are ordered by different anchors due to the ordering delay. Intuitively, B_v and B_a are more likely to be ordered by the same anchor if their anchors are far apart, i.e., there are more rounds for the anchors to construct paths to them. In this case, the number of liveness attackers f_l can affect the probabilities of $Pr(\Phi_3)$ and $Pr(\Phi_4)$ because liveness attackers can prolong the selection of a valid anchor. To conclude, $Pr(\Phi_3)$ and $Pr(\Phi_4)$ should be negatively related to f_l, and we respectively let $Pr(\Phi_3) = \tau_1$ and $Pr(\Phi_4) = \tau_2$. It is worth noting that since the attacking block B_a is created later than the victim block B_v in the attacking game, we have $\tau_1 < \tau_2$ according to **Prop4**. Additionally, as we will discuss below, τ_2 varies with distinct attacking strategies as they somewhat delay the creation of B_a.

B.2. Analysis on the Fissure Attack

Without loss of generality, we assume the victim block B_v^i and the attacking block B_a^i are created in the same round r_i. We assume all honest nodes have both B_v^i and B_a^i in the connections of their blocks belonging to round r_{i+1}[14].

Uneven connections. When performing the fissure attack, $\mathcal{A}$ create their blocks $\sigma_a^{i+1} = \{B_{a_1}^{i+1}, \cdots, B_{a_{f_a}}^{i+1}\}$ in round r_{i+1} without connecting B_v^i. This will lead to *uneven connections* between B_v^i and B_a^i. Let $\mathcal{C}_a^{i+1}$ denote the set of blocks in round r_{i+1} that connect B_a^i, and $\mathcal{C}_v^{i+1}$ denote the set of blocks in round r_{i+1} that connect B_v^i. Under the assumption that all honest nodes' blocks connect both B_v^i and B_a^i, we have $\mathcal{C}_a^{i+1} = \mathcal{C}_v^{i+1} \bigcup \sigma_a^{i+1}$. In this case, we have (1) if a block from σ_a^{i+1} is the first traversed block of round r_{i+1}, then B_a^i is ordered before B_v^i because B_a^i is connected by the block but B_v^i is not; (2) if, otherwise, a block from $\mathcal{C}_v^{i+1}$ is the first traversed block of round r_{i+1}, then the order between B_a^i and B_v^i depends on the block digest (i.e., 50% probability that B_a^i is ordered before B_v^i according to **Prop2**). To conclude, let $\mathcal{F}_0^{fis}$ denote the probability of ordering B_a^i before B_v^i by the blocks in r_{i+1}, we can get:

$$\mathcal{F}_0^{fis} = \frac{|\sigma_a^{i+1}|}{|\mathcal{C}_a^{i+1}|} \cdot 1 + \frac{|\mathcal{C}_v^{i+1}|}{|\mathcal{C}_a^{i+1}|} \cdot \frac{1}{2} = \frac{1}{2} + \frac{f_a}{2(n - f_l)} \tag{1}$$

Cumulative influence. In the fissure attack, $\mathcal{A}$ keep excluding the blocks of the victim from the connections of their blocks. Therefore, it has a *cumulative influence* in this attacking game. To elaborate, blocks in $\mathcal{C}_a^{i+1}$ and blocks in $\mathcal{C}_v^{i+1}$ are also connected with the uneven numbers of blocks in round r_{i+2}, denoted by $\mathcal{C}_{\sigma_a}^{i+2}$ and $\mathcal{C}_{\sigma_v}^{i+2}$ respectively. Due to the influence of the fissure attack, we have $|\mathcal{C}_{\sigma_a}^{i+2}| \geq |\mathcal{C}_{\sigma_v}^{i+2}|$. The difference in connections between N_v and N_a *exists in every subsequent round until the next anchor round*. Therefore, the cumulative probability that B_a^i is ordered before B_v^i via the fissure attack can be defined as $\mathcal{P}^{fis}(\nabla_r)$ that is related to ∇_r. Furthermore, we observe that as blocks get farther away from B_v^i and B_a^i, the influence of these attacking blocks will diminish. This is because as the number of rounds increases, B_v^i will be more likely to be traversed *indirectly* by the blocks from $\mathcal{A}$ (e.g., in Figure 4, B_v^i can be traversed by B_a^{i+2} via the path $\langle B_a^{i+2}, B_2^{i+1}, B_v^i \rangle$). To this end, we introduce a decay factor $\gamma \in (0, 1)$ to model such a diminished influence. Consequently, for each ∇_r, $\mathcal{P}^{fis}(\nabla_r)$ can be calculated by:

$$\mathcal{P}^{fis}(\nabla_r) = \sum_{k=1}^{\nabla_r} \gamma^{k-1} \mathcal{F}_0^{fis} \tag{2}$$

Impact of attacks across anchors. When evaluating the probability that $\mathcal{A}$ win the game under the fissure attack $\mathcal{P}_{Adv}^{fis}(\nabla_r)$, we need to consider the scenarios where the attack is across anchors, i.e., the conditions $\Phi_1 - \Phi_4$ as mentioned in Appendix B.1. Moreover, since the fissure attack excludes blocks from the victim nodes, $\mathcal{A}$ may require more time to collect enough blocks (i.e., $n - t$ blocks) to move to the next round. In this case, the attacking block will likely experience an ordering delay. Therefore, we use τ_2^{fis} to denote the probability of Φ_4 under the fissure attack, where $\tau_2^{fis} > \tau_2$. Specifically, $\mathcal{P}_{Adv}^{fis}(\nabla_r)$ can be calculated as follows:

$$\begin{aligned}
\mathcal{P}_{Adv}^{fis}(\nabla_r) &= \begin{cases} 0, \text{ if } \Phi_1 \text{ or } \Phi_4 \text{ satisfies,} \\ 1, \text{ if } \Phi_2 \text{ or } \Phi_3 \text{ satisfies,} \\ \mathcal{P}^{fis}(\nabla_r), \text{ Otherwise.} \end{cases} \\
&= \frac{1}{\mathcal{W}n} \cdot 0 + \tau_2^{fis} \cdot 0 + \frac{1}{\mathcal{W}n} \cdot 1 + \tau_1 \cdot 1 \\
&\quad + (1 - \frac{1}{\mathcal{W}n} - \frac{1}{\mathcal{W}n} - \tau_1 - \tau_2^{fis}) \cdot \mathcal{P}^{fis}(\nabla_r) \\
&= \underbrace{\frac{1}{\mathcal{W}n} + \tau_1}_{\text{replaced by } \alpha} + \underbrace{(1 - \frac{2}{\mathcal{W}n} - \tau_1 - \tau_2^{fis})}_{\text{replaced by } \beta^{fis}} \cdot \mathcal{P}^{fis}(\nabla_r) \\
&= \alpha + \beta^{fis} \cdot \mathcal{P}^{fis}(\nabla_r)
\end{aligned} \tag{3}$$

In the above equation, we introduce two factors $\alpha = \frac{1}{\mathcal{W}n} + \tau_1$ and $\beta^{fis} = 1 - \frac{2}{\mathcal{W}n} - \tau_1 - \tau_2^{fis}$ to simplify the expression.

Expected attacking success rate. $\mathcal{P}_{Adv}^{fis}(\nabla_r)$ indicates that $\mathcal{A}$ win the attacking game when B_v is ordered and committed by a subsequent anchor after ∇_r rounds. We now calculate the expected probability that $\mathcal{A}$ win the attacking game. Let $P(\nabla_r)$ denote the distribution function of ∇_r. Note that since liveness attackers do not create blocks, both B_v and B_a can only be ordered by an anchor that is not from the liveness attackers. Eventually, the probability that B_v and B_a require $\mathcal{R}$ rounds to be ordered is:

$$P(\nabla_r = \mathcal{R}) = \frac{1}{\mathcal{W}} (\frac{n - f_l}{n}) (\frac{f_l}{n})^{\lfloor \mathcal{R}/\mathcal{W} \rfloor} \tag{4}$$

14. In practice, B_v^i and B_a^i can be connected by distinct honest nodes' blocks in round r_{i+1}. However, the number of honest nodes connecting B_v^i is close to the number of honest nodes connecting B_a^i, which determines the calculation of the attacking probability. Therefore, we can simplify it with this assumption.

Note that:

$$\lim_{m\to\infty}\sum_{\mathcal{R}=1}^{m} P(\nabla_r=\mathcal{R})=1 \tag{5}$$

When combining Equation (1)-(5), we can get the expected probability that $\mathcal{A}$ win the attacking game $\mathbb{E}[\mathcal{P}_{Adv}^{fis}(\nabla_r)]$ by:

$$\begin{aligned}
&\mathbb{E}[\mathcal{P}_{Adv}^{fis}(\nabla_r)]=\sum_{\mathcal{R}=1}^{\infty} P(\nabla_r=\mathcal{R})\cdot\mathcal{P}_{Adv}^{fis}(\nabla_r)\\
=&\sum_{\mathcal{R}=1}^{\infty} P(\nabla_r=\mathcal{R})\cdot\alpha+\sum_{\mathcal{R}=1}^{\infty} P(\nabla_r=\mathcal{R})\cdot\beta^{fis}\cdot\mathcal{P}^{fis}(\nabla_r)\\
=&\alpha+\beta^{fis}\sum_{\mathcal{R}=1}^{\infty}\frac{1}{\mathcal{W}}(\frac{n-f_l}{n})(\frac{f_l}{n})^{\lfloor\mathcal{R}/\mathcal{W}\rfloor}\cdot\sum_{k=1}^{\mathcal{R}}\gamma^{k-1}\mathcal{F}_0^{fis}\\
=&\alpha+\beta\cdot\frac{1}{\mathcal{W}}\cdot\frac{(n-f_l)}{n}\cdot\mathcal{F}_0^{fis}\cdot\sum_{\mathcal{R}=1}^{\infty}(\frac{f_l}{n})^{\lfloor\mathcal{R}/\mathcal{W}\rfloor}\cdot\sum_{k=1}^{\mathcal{R}}\gamma^{k-1}\\
=&\alpha+\frac{(n+f_a-f_l)\beta^{fis}}{2\mathcal{W}n}\sum_{\mathcal{R}=1}^{\infty}(\frac{f_l}{n})^{\lfloor\mathcal{R}/\mathcal{W}\rfloor}\sum_{k=1}^{\mathcal{R}}\gamma^{k-1}
\end{aligned} \tag{6}$$

B.3. Analysis on Speculative Attack

One-shot influence. Unlike the fissure attack, the speculative attack has a *one-shot influence* in this attacking game. Specifically, the high-level idea of the speculative attack is to create an attacking block B_a with the largest possible digest, which is only decided by the round of B_a and is independent of the subsequent rounds. In this case, the probability that $\mathcal{A}$ wins the game under the speculative attack is determined by how possible N_a is to create a block with a larger digest than that of B_v in a round. Recall that each node has f_w transaction batches when creating a new block (cf. **Prop3**), and therefore N_a can create $\mathcal{M}_{f_w}$ blocks in each round, where $\mathcal{M}_{f_w}$ is the number of permutations of the f_w transaction batches. Recall that the probability that each block created by N_a has a larger digest than that of B_v is 50% (cf. **Prop2**). If we define $\mathcal{F}^{pkm}(f_w)$ as the probability that N_a can create at least one block with a larger digest than the digest of B_v, then we have:

$$\mathcal{F}^{pkm}(f_w)=1-\frac{1}{2^{\mathcal{M}_{f_w}}} \tag{7}$$

Impact of attacks across anchors. Similarly, we need to consider the speculative attack across anchors. Since the speculative attack needs to construct multiple blocks to get an attacking block with the largest digest, it takes N_a more time to get a new block. In this case, the attacking block will likely experience an ordering delay. Therefore, we use τ_2^{pkm} to denote the probability of Φ_4 under the speculative attack, where $\tau_2^{pkm} > \tau_2$. Eventually, we can calculate the probability that $\mathcal{A}$ win the attacking game under the speculative attack $\mathcal{P}_{Adv}^{pkm}$ by :

$$\begin{aligned}
\mathcal{P}_{Adv}^{pkm}=&\begin{cases}0, \text{ if } \Phi_1 \text{ or } \Phi_4 \text{ satisfies,}\\ 1, \text{ if } \Phi_2 \text{ or } \Phi_3 \text{ satisfies,}\\ \mathcal{F}^{pkm}(f_w), \text{ Otherwise.}\end{cases}\\
=&\frac{1}{\mathcal{W}n}\cdot 0+\tau_2^{pkm}\cdot 0+\frac{1}{\mathcal{W}n}\cdot 1+\tau_1\cdot 1\\
&+(1-\frac{1}{\mathcal{W}n}-\frac{1}{\mathcal{W}n}-\tau_1-\tau_2^{pkm})\cdot\mathcal{F}^{pkm}(f_w)\\
=&\underbrace{\frac{1}{\mathcal{W}n}+\tau_1}_{\text{replaced by }\alpha}+\underbrace{(1-\frac{2}{\mathcal{W}n}-\tau_1-\tau_2^{pkm})}_{\text{replaced by }\beta^{pkm}}\cdot\mathcal{F}^{pkm}(f_w)\\
=&\alpha+\beta^{pkm}\cdot\mathcal{F}^{pkm}(f_w)
\end{aligned} \tag{8}$$

Similarly, we introduce factors $\alpha=\frac{1}{\mathcal{W}n}+\tau_1$ and $\beta^{pkm}=1-\frac{2}{\mathcal{W}n}-\tau_1-\tau_2^{pkm}$ to simplify the expression.

Expected attacking success rate. Similarly, we evaluate the expected attacking success rate of the speculative attack $\mathbb{E}[\mathcal{P}_{Adv}^{pkm}(\nabla_r)]$ by considering that B_v (or B_a) is ordered by the anchor after distinct numbers of rounds ∇_r. Note that since $\mathcal{P}_{Adv}^{pkm}$ is independent of ∇_r, we have $\mathcal{P}_{Adv}^{pkm}=\mathcal{P}_{Adv}^{pkm}(\nabla_r)$. Consequently, when combining Equation (4)-(5), (7)-(8), we can evaluate $\mathbb{E}[\mathcal{P}_{Adv}^{pkm}(\nabla_r)]$ by:

$$\begin{aligned}
&\mathbb{E}[\mathcal{P}_{Adv}^{pkm}(\nabla_r)]\\
=&\sum_{\mathcal{R}=1}^{\infty} P(\nabla_r=\mathcal{R})\cdot\mathcal{P}_{Adv}^{pkm}(\nabla_r)=1\cdot\mathcal{P}_{Adv}^{pkm}\\
=&\alpha+\beta^{pkm}(1-\frac{1}{2^{\mathcal{M}_{f_w}}})
\end{aligned} \tag{9}$$

B.4. Analysis on Sluggish Attack

When performing the sluggish attack, the designated frontrunning attacker N_a strategically delays the process of the block creation. As we introduced in Section 5.1, one of the approaches to slow down is to multiply the timeout of creating new blocks by an attacking factor t_s. With an appropriate t_s, N_a can be just working in a round smaller than the round of the victim block B_v, in which N_a can create the attacking block B_a with a round number smaller than B_v and successfully front-run B_v. Therefore, we define the probability that N_a can create B_a in a smaller round than B_v as a function $\mathcal{F}^{slg}(t_s)$ that is related to t_s.

Sluggish dilemma. With the sluggish attack, if, fortunately, the victim block B_v and the attacking block B_a are ordered and committed by the same anchor, then B_a must be ordered before B_v, i.e., $\mathcal{A}$ win the game. However, from our evaluation, we observe that there is a *sluggish dilemma* in the sluggish attack due to the ordering delay. Specifically, in the DAG-based SMR protocol, a node can move to the next round r_{i+1} once it receives $n-f$ blocks from the current round r_i, and its newly created block in r_{i+1} will connect the received $n-f$ blocks but not the blocks of r_i that it has not received. In this case, if the designated frontrunning attacker N_a is too slow to make its attacking block B_a connected by blocks of the subsequent rounds, then B_a will experience an ordering delay, as illustrated in Figure 13(a). On the other hand, if N_a moves too fast (or at a normal speed as honest nodes), then N_a may be working

in a larger round than (or the same round as) the victim N_v after it monitors B_v, which will make the attack failed.

Impact of attacks across anchors. Because of the sluggish dilemma, $\mathcal{A}$ are more likely to experience an ordering delay when performing the sluggish attacking strategy. Therefore, we use τ_2^{slg} to denote the probability of Φ_4 under the speculative attack, where $\tau_2^{slg} > \tau_2$. Let $\mathcal{P}_{Adv}^{slg}$ denote the probability that $\mathcal{A}$ win the attacking game under the sluggish attack. Similarly, when considering the attack crossing anchors, we have:

$$\begin{aligned}\mathcal{P}_{Adv}^{slg} = &\begin{cases} 0, \text{ if } \Phi_1 \text{ or } \Phi_4 \text{ satisfies,} \\ 1, \text{ if } \Phi_2 \text{ or } \Phi_3 \text{ satisfies,} \\ \mathcal{F}^{slg}(t_s), \text{ Otherwise.} \end{cases} \\ =&\frac{1}{\mathcal{W}n}\cdot 0 + \tau_2^{slg}\cdot 0 + \frac{1}{\mathcal{W}n}\cdot 1 + \tau_1 \cdot 1 \\ &+ (1 - \frac{1}{\mathcal{W}n} - \frac{1}{\mathcal{W}n} - \tau_1 - \tau_2^{slg})\cdot \mathcal{F}^{slg}(t_s) \\ =&\underbrace{\frac{1}{\mathcal{W}n} + \tau_1}_{\text{replaced by } \alpha} + \underbrace{(1 - \frac{2}{\mathcal{W}n} - \tau_1 - \tau_2^{slg})}_{\text{replaced by } \beta^{slg}} \cdot \mathcal{F}^{slg}(t_s) \\ =&\alpha + \beta^{slg}\mathcal{F}^{slg}(t_s)\end{aligned} \tag{10}$$

Similarly, we introduce factors $\alpha = \frac{1}{\mathcal{W}n} + \tau_1$ and $\beta^{slg} = 1 - \frac{2}{\mathcal{W}n} - \tau_1 - \tau_2^{slg}$ to simplify the expression.

Expected attacking success rate. Let $\mathbb{E}[\mathcal{P}_{Adv}^{slg}(\nabla_r)]$ denote the expected success rate of the sluggish attack. Similar to the analysis of the speculative attack, $\mathcal{P}_{Adv}^{slg}$ is independent of ∇_r and therefore $\mathcal{P}_{Adv}^{slg} = \mathcal{P}_{Adv}^{slg}(\nabla_r)$. Eventually, when combining Equation (4)-(5), (10), we have:

$$\begin{aligned}&\mathbb{E}[\mathcal{P}_{Adv}^{slg}(\nabla_r)] \\ =&\sum_{\mathcal{R}=1}^{\infty} P(\nabla_r = \mathcal{R}) \cdot \mathcal{P}_{Adv}^{slg}(\nabla_r) = 1 \cdot \mathcal{P}_{Adv}^{slg} \\ =&\alpha + \beta^{slg} \cdot \mathcal{F}^{slg}(t_s)\end{aligned} \tag{11}$$

Appendix C. Evaluation and Analysis on the Degradation of the Fissure Attack

In Section 5.3, we observe the degradation of the fissure attack on Tusk. We speculate this is because the delegated frontrunning attacker needs more time to move to the next round, in which honest nodes are less likely to connect the attacking block. Specifically, in the Tusk protocol, nodes move to the next round once they receive and can connect $n - t$ blocks from the current round, where $t = \lfloor \frac{n-1}{3} \rfloor$ is the Byzantine fault tolerance of the protocol. Since the frontrunning attackers do not connect blocks from the victim node, they need to receive more blocks (i.e., $n-t+1 > n-t$) and thereby require more time before they can move to the next round and create a new block. If $f_a + f_l \leq t$, the victim node and other honest nodes can process normally without waiting for the slow blocks from the attackers. In this case, honest nodes will connect the victim block but not the attacking block, making the attacking block more likely to be ordered after the victim block according to the connection priority (Observation 4.2.1).

To verify this hypothesis, we further evaluate the fissure attack with different n, f_a, and f_l satisfying $f_a + f_l = \lfloor \frac{n-1}{3} \rfloor$. The results are shown in Table 3. We find that the fissure attack on Tusk indeed degrades the ASR when f_a is small but will perform effectively with f_a increasing. For the Bullshark protocol that delays all nodes' movement as analyzed in Section 5.4, the fissure attack works effectively in most cases.

TABLE 3: Attack success rate (%) with varying parameters n, f_a, and f_l satisfying $f_a + f_l = \lfloor \frac{n-1}{3} \rfloor$.

n	$[f_a, f_l]$	Bullshark		Tusk	
		Baseline	Fissure	Baseline	Fissure
10	[1, 2]	47.32	51.84	40.1	24.37
	[2, 1]	48.41	58.09	37.75	31.37
13	[1, 3]	50.59	50.12	42.45	22.32
	[2, 2]	52.5	53.13	40.2	34.39
	[3, 1]	50.95	58.69	34.39	39.71
16	[1, 4]	47.36	47.2	39.68	18.08
	[2, 3]	45.91	53.95	38.06	32.43
	[3, 2]	49.47	57.43	34.95	42.7
	[4, 1]	46.36	59.1	33.98	40.08
19	[1, 5]	49.5	37.74	43.12	18.37
	[3, 3]	45.89	57.2	41.5	39.32
	[5, 1]	47.76	63.06	40.7	46.81
25	[1, 7]	49.52	38.69	44.22	21.56
	[4, 4]	50.28	58.46	37.31	46.77
	[7, 1]	49.13	63.67	36.68	48.39