Title: Overcoming Gas Estimation Challenges in NFT Transactions with Dynamic Arrays: A Comprehensive Analysis and Recommendations

Introduction:

Non-Fungible Tokens (NFTs) have gained significant traction in recent years, becoming a pivotal component in the world of decentralized applications. However, as the popularity of NFTs grows, so do the complexities associated with managing them on the Ethereum network. One such complexity is accurately estimating gas usage for NFT transactions involving dynamic arrays, which is essential for ensuring smooth and efficient execution of transactions on the blockchain.

Inaccurate gas estimation can lead to out-of-gas errors, wasted resources, and transaction delays, ultimately impacting the user experience and overall network performance. Despite the availability of various gas estimation methods, such as web3.eth.estimateGas, eth_estimateGas JSON-RPC method, eth_call JSON-RPC method with gas field, static analysis tools, and gas estimation services, challenges persist in estimating gas usage for NFT transactions with dynamic arrays.

This paper aims to provide a comprehensive analysis of the factors contributing to the challenges in gas estimation for NFT transactions involving dynamic arrays, such as state-dependent conditions, transaction complexity, variable-length loops, and interactions with other smart contracts. We will evaluate the performance of popular gas estimation methods in the context of these challenges, and through a case study, we will illustrate the impact of these challenges on real-world NFT transactions.

Finally, we will present a set of recommendations and improvements to address these challenges, including implementing safety margins, developing adaptive gas estimation algorithms, using testing and monitoring tools, and following best practices for NFT contract developers. By adopting these strategies, developers can enhance the accuracy and reliability of gas estimation, paving the way for more efficient and scalable decentralized applications involving NFTs and dynamic arrays.

1.1. Background on gas estimation methods:

Gas estimation is an essential aspect of executing transactions on the Ethereum network. Accurate gas estimation helps prevent out-of-gas errors, minimizes wasted gas, and ensures smooth transaction processing. In this section, we provide an overview of the various gas estimation methods used in the Ethereum ecosystem:

a) web3.eth.estimateGas: This method is provided by the Web3.js library and is widely used by developers. It simulates the transaction execution in the local Ethereum Virtual Machine (EVM) and observes the gas consumption to estimate the gas limit required for a successful transaction. The accuracy of this method can vary depending on the transaction's complexity and the contract's state.

b) eth_estimateGas JSON-RPC method: Ethereum clients, such as Geth and OpenEthereum, expose a JSON-RPC API that developers can use to interact with the Ethereum network. The eth_estimateGas method is part of this API and works similarly to web3.eth.estimateGas. It simulates transaction execution in the EVM to estimate the gas usage. Different Ethereum clients may have slightly varying implementations, which can lead to minor discrepancies in gas estimation.

c) eth_call JSON-RPC method with gas field: This method involves using the eth_call method, which simulates a transaction's execution without actually sending it. The gas field in the response can provide an estimate of the gas limit. However, this method might not be as accurate as the eth_estimateGas method, as it doesn't specifically focus on estimating gas usage.

d) Static analysis tools: Some static analysis tools can provide gas usage estimates for smart contracts. These tools analyze the contract's bytecode and calculate gas usage based on the opcodes executed. However, this method might not account for dynamic gas usage resulting from state-dependent conditions or complex transaction interactions. Examples of such tools include Remix and MythX.

e) Gas estimation services: Some third-party services provide gas estimation based on the current state of the Ethereum network. These services analyze pending and mined transactions to suggest appropriate gas prices and gas limits. Examples include ETH Gas Station and Gas Now. While these services can help you set appropriate gas prices for your transactions, they might not be as accurate in estimating gas limits for specific transaction types.

Each of these methods has its own approach to estimating gas usage for transactions, and their accuracy can vary depending on various factors such as transaction complexity, contract state, and interaction with other contracts. Understanding the strengths and limitations of each method can help developers choose the most suitable approach for their specific use case.

1.2. The importance of accurate gas estimation:

Accurate gas estimation is crucial for the smooth functioning of transactions on the Ethereum network. It plays a vital role in ensuring that transactions are executed efficiently and without unnecessary delays. In this section, we emphasize the significance of accurate gas estimation and discuss the consequences of inaccurate gas estimation.

a) Out-of-gas errors: If a transaction's gas limit is set too low, the transaction may run out of gas during execution, causing it to fail. Out-of-gas errors can be particularly problematic for users, as they not only result in failed transactions but also lead

to the loss of the gas fees paid to initiate the transaction. Accurate gas estimation can help prevent out-of-gas errors by ensuring that the gas limit is set high enough to cover the transaction's gas requirements.

b) Wasted resources: Overestimating the gas limit for a transaction can result in wasted resources. Although any unused gas is refunded to the sender, a higher gas limit may cause miners to deprioritize the transaction due to the perceived higher computational requirements. Consequently, the transaction may take longer to be included in a block, and the sender may end up paying higher fees than necessary.

c) Transaction delays: Inaccurate gas estimation can lead to transaction delays. Underestimating gas requirements may result in out-of-gas errors and necessitate resubmitting the transaction with a higher gas limit. Overestimating gas usage can cause miners to deprioritize the transaction, increasing the time it takes to be included in a block. Both situations negatively impact user experience and network performance.

d) Network congestion: Inaccurate gas estimation can contribute to network congestion. When a large number of transactions with overestimated or underestimated gas limits are submitted to the network, it can lead to an inefficient allocation of resources and increased congestion. Accurate gas estimation helps optimize resource allocation, reducing network congestion and improving overall performance.

In conclusion, accurate gas estimation is essential for maintaining the smooth functioning of the Ethereum network. It helps prevent out-of-gas errors, minimizes wasted resources, reduces transaction delays, and alleviates network congestion. Ensuring accurate gas estimation can greatly enhance the user experience and overall network performance.

1.3. NFTs and dynamic arrays:

Non-Fungible Tokens (NFTs) have gained significant popularity in the Ethereum ecosystem due to their unique properties and various use cases, such as digital art, collectibles, and virtual goods. In this section, we introduce NFTs and discuss their relevance in the context of gas estimation.

NFTs are individual tokens representing ownership of unique digital assets. They are typically implemented using the ERC-721 or ERC-1155 token standards, which define a set of functions and events for managing the creation, transfer, and ownership of NFTs on the Ethereum blockchain.

To manage NFT ownership, smart contracts often use dynamic arrays to track the NFTs a user owns. These arrays can grow or shrink as NFTs are transferred between users. While dynamic arrays provide an efficient means of tracking NFT ownership, they also introduce complexity and state-dependent conditions, which can make accurate gas estimation a challenging task.

Some factors contributing to this challenge include:

a) State-dependent conditions: The gas usage for a transaction involving NFTs and dynamic arrays can vary depending on the current state of the contract's storage. For example, the gas cost for adding or removing an NFT from a user's dynamic array may depend on the array's current size or the position of the NFT within the array. These state-dependent conditions can make it difficult for gas estimation algorithms to predict the correct gas limit.

b) Transaction complexity: NFT transactions can involve complex interactions between multiple smart contracts, such as marketplaces, escrow services, or royalty distribution systems. These complex interactions can introduce additional gas costs and make it harder for gas estimation algorithms to accurately estimate the total gas usage.

c) Variable-length loops: Some NFT-related functions may include loops that iterate over a user's dynamic array of owned NFTs. The number of loop iterations can vary depending on the array's size, making the gas usage for such functions dependent on the contract's state. This can result in gas estimation challenges, as the gas usage may change with each new NFT added to or removed from a user's array.

In conclusion, NFT transactions involving dynamic arrays can introduce complexity and state-dependent conditions that make accurate gas estimation a difficult task. Understanding these challenges is crucial for developing and refining gas estimation methods that can better handle the intricacies of NFT transactions.

2.1. State-dependent conditions:

State-dependent conditions are a significant factor contributing to inaccuracies in gas estimation for NFT transactions involving dynamic arrays. In this section, we discuss how the gas usage can vary depending on the current state of the contract's storage, making it difficult for estimation algorithms to predict the correct gas limit.

When dealing with NFTs and dynamic arrays, the gas cost of executing a function may change based on the current state of the contract's storage, such as the size of the array, the position of the NFT within the array, or the existence of certain conditions. This variation can make it challenging for gas estimation algorithms to provide accurate estimates.

Examples of state-dependent conditions include:

a) Array size: Functions that modify dynamic arrays, such as adding or removing NFTs, can have gas costs that depend on the array's current size. As the array grows, the gas cost for inserting or removing elements may increase, leading to higher

gas consumption. Gas estimation algorithms may struggle to accurately predict gas usage in these scenarios, especially when dealing with large arrays.

b) NFT position: In some cases, the gas usage for a function may depend on the position of the NFT within the dynamic array. For example, removing an NFT from the beginning of the array may require more gas than removing it from the end, as the contract needs to shift the elements in the array to fill the gap. Gas estimation algorithms may not account for these positional differences, resulting in inaccurate estimates.

c) Conditional gas costs: Some functions in NFT contracts may have conditional gas costs, meaning that the gas usage depends on specific conditions being met. For example, a function might require additional gas if a certain NFT has a particular attribute or if the NFT is part of a specific collection. These conditional gas costs can be difficult for gas estimation algorithms to predict, leading to inaccuracies in the estimates.

In conclusion, state-dependent conditions in NFT transactions with dynamic arrays can make accurate gas estimation challenging. To improve the accuracy of gas estimates, algorithms need to consider the various factors that may affect gas usage and adapt to the current state of the contract's storage. This may involve incorporating more sophisticated techniques, such as static or dynamic analysis, to better predict gas consumption in the presence of state-dependent conditions.

2.2. Transaction complexity:

Transaction complexity plays a crucial role in gas estimation, particularly for NFT transactions with dynamic arrays. In this section, we examine the impact of transaction complexity on gas estimation and explain how it can pose challenges for gas estimation algorithms, leading to inaccurate estimates.

Complex transactions, especially those involving multiple smart contract interactions, can have varying gas requirements due to the combined effects of individual contract functions, events, and data storage operations. These interactions can make it difficult for gas estimation algorithms to provide accurate estimates for several reasons:

a) Interdependent gas costs: When multiple smart contracts interact, the gas cost of one function can be influenced by the gas consumption of other functions in the same transaction. This interdependence can make it difficult for gas estimation algorithms to accurately predict the total gas usage, as they need to consider the combined gas costs of all interacting functions.

b) Unpredictable execution paths: Complex transactions can involve conditional execution paths, where the flow of execution depends on certain conditions being met. For example, an NFT marketplace contract might have different execution paths for placing a bid, buying an NFT outright, or canceling a bid. Gas estimation algorithms may struggle to account for these unpredictable execution paths, leading to inaccuracies in the estimates.

c) Nested function calls: NFT transactions with dynamic arrays can include nested function calls, where one function calls another function within the same or a different contract. These nested calls can introduce additional gas costs and execution complexity, making it challenging for gas estimation algorithms to accurately predict the total gas usage.

d) External contract interactions: In some cases, NFT transactions might interact with external contracts, such as oracles, escrow services, or royalty distribution systems. The gas consumption of these external interactions can be difficult to predict, as it depends on factors that are outside the control of the NFT contract, such as the implementation and state of the external contract.

In conclusion, transaction complexity can significantly impact gas estimation for NFT transactions with dynamic arrays. Accurate gas estimation requires algorithms that can account for the various factors contributing to complexity, such as interdependent gas costs, unpredictable execution paths, nested function calls, and external contract interactions. Developing more sophisticated gas estimation techniques that can handle transaction complexity will be crucial for ensuring accurate estimates and reducing out-of-gas errors in complex NFT transactions.

2.3. Variable-length loops:

Variable-length loops present a significant challenge for gas estimation in NFT transactions with dynamic arrays. In this section, we analyze the role of variable-length loops in causing inaccuracies in gas estimation and explain how they can contribute to out-of-gas errors.

Functions in NFT contracts may include loops that iterate over dynamic arrays, with the number of iterations depending on the current state of the contract. The gas usage of these functions can vary based on the size of the array or the number of NFTs owned by a user, making it difficult for gas estimation algorithms to provide accurate estimates. Several factors contribute to the challenges posed by variable-length loops:

a) Dynamic gas usage: The gas consumption of a loop depends on the number of iterations it performs, which can change based on the current state of the contract. This dynamic gas usage can make it difficult for gas estimation algorithms to predict the correct gas limit, as they need to consider the varying gas costs associated with different loop lengths.

b) Nested loops: In some cases, NFT transactions with dynamic arrays may involve nested loops, where one loop is contained within another. Nested loops can increase the complexity of gas estimation, as they can lead to exponential

growth in the number of iterations, making it challenging for estimation algorithms to predict the total gas usage accurately.

c) Unbounded loops: Certain functions in NFT contracts may include loops without a predetermined maximum number of iterations. These unbounded loops can be especially problematic for gas estimation algorithms, as they may result in an unpredictable gas consumption that could significantly exceed the estimated gas limit, causing out-of-gas errors.

d) Data-dependent loops: Some loops in NFT contracts may have a variable number of iterations based on the data stored in the contract, such as the attributes of NFTs or the relationships between different NFT collections. Estimating the gas consumption of data-dependent loops can be challenging, as gas estimation algorithms need to consider the gas costs associated with various data conditions.

In conclusion, variable-length loops can cause inaccuracies in gas estimation for NFT transactions with dynamic arrays, contributing to out-of-gas errors. To improve the accuracy of gas estimates, algorithms need to consider the factors contributing to the challenges posed by variable-length loops, such as dynamic gas usage, nested loops, unbounded loops, and data-dependent loops. Developing more advanced gas estimation techniques that can handle the complexities introduced by variable-length loops will be crucial for reducing out-of-gas errors in NFT transactions.

2.4. Interaction with other smart contracts:

Interactions between NFT contracts and other smart contracts can introduce additional complexity and uncertainty in gas estimation. In this section, we discuss how gas usage might be influenced by the behavior of external contracts and how this can result in inaccurate gas estimates.

NFT transactions often involve interactions with other smart contracts on the Ethereum network, such as marketplaces, escrow services, or royalty distribution systems. These external contracts can influence gas usage in various ways:

a) Varying implementation: Different external contracts might have varying implementations, which can lead to different gas costs for the same function. Gas estimation algorithms need to consider these variations in implementation when estimating gas usage, as the gas costs might not be the same for all external contract interactions.

b) State-dependent behavior: The behavior of external contracts might also depend on their current state, which can impact the gas usage of the NFT transaction. For example, an escrow contract might require more gas to execute a function if it is in a specific state, such as when a dispute is ongoing. This state-dependent behavior can introduce additional uncertainty in gas estimation, making it difficult for algorithms to provide accurate estimates.

c) Unpredictable gas consumption: Some external contracts might have functions with unpredictable gas consumption, such as those involving complex calculations or variable-length loops. These functions can introduce additional uncertainty in gas estimation, as the gas usage might be difficult to predict without knowing the exact input parameters or the state of the external contract.

d) Fallback functions: Interactions with external contracts might trigger fallback functions, which are executed when a contract receives Ether without a specific function being called. Fallback functions can have varying gas costs, depending on the implementation and the state of the external contract, which can introduce additional complexity in gas estimation.

In conclusion, interactions between NFT contracts and other smart contracts can further complicate gas estimation, as gas usage might be influenced by the behavior of external contracts. To improve the accuracy of gas estimates, algorithms need to account for the various factors introduced by external contract interactions, such as varying implementation, state-dependent behavior, unpredictable gas consumption, and fallback functions. Developing more sophisticated gas estimation techniques that can handle the complexities of interacting with other smart contracts will be essential for providing accurate estimates and reducing out-of-gas errors in NFT transactions.

3.1. web3.eth.estimateGas:

web3.eth.estimateGas is a popular method for estimating gas usage in Ethereum transactions, including NFT transactions with dynamic arrays. In this section, we evaluate the performance of web3.eth.estimateGas, discussing its strengths and limitations, specifically focusing on its accuracy in handling transactions involving state-dependent conditions, transaction complexity, variable-length loops, and interactions with other smart contracts.

Strengths:

Ease of use: web3.eth.estimateGas is easily accessible through the widely-used web3.js library, making it a convenient option for developers working with Ethereum applications.

Broad applicability: web3.eth.estimateGas can be used for a wide range of transaction types, including NFT transactions with dynamic arrays, making it a versatile method for gas estimation.

Limitations:

State-dependent conditions: web3.eth.estimateGas might struggle to provide accurate gas estimates for transactions involving state-dependent conditions, as it may not have complete information about the current state of the contract's storage.

Transaction complexity: web3.eth.estimateGas may not be well-suited for handling complex transactions, particularly those involving multiple smart contract interactions, conditional execution paths, or nested function calls, leading to potential inaccuracies in gas estimation.

Variable-length loops: This method might not accurately predict the gas usage for functions with variable-length loops, as it might not have sufficient information about the number of iterations or the dynamic behavior of the loops.

Interactions with other smart contracts: web3.eth.estimateGas might not account for the gas usage implications of interacting with external contracts, especially when those interactions involve varying implementations, state-dependent behavior, unpredictable gas consumption, or fallback functions.

In conclusion, web3.eth.estimateGas has its strengths in ease of use and broad applicability, but it may not be the most accurate method for estimating gas usage in NFT transactions with dynamic arrays that involve state-dependent conditions, transaction complexity, variable-length loops, or interactions with other smart contracts. Developers may need to employ additional techniques, such as static analysis or adaptive gas estimation algorithms, to improve the accuracy of gas estimates in these scenarios.

3.2. eth_estimateGas JSON-RPC method:

The eth_estimateGas JSON-RPC method is another popular technique used to estimate gas usage in Ethereum transactions, including NFT transactions with dynamic arrays. In this section, we assess the eth_estimateGas method, examining its implementation in different Ethereum clients and analyzing its accuracy in handling various complexities associated with these transactions. We will also discuss the potential factors contributing to its limitations.

Implementation in Ethereum clients:

The eth_estimateGas method is part of the Ethereum JSON-RPC API, which is implemented in various Ethereum clients, such as Geth, OpenEthereum, and Besu. The specific implementation details of eth_estimateGas may vary between clients, potentially leading to slight differences in gas estimates. However, the general approach involves simulating the transaction execution in a local environment to estimate gas consumption.

Accuracy in handling complexities:

State-dependent conditions: eth_estimateGas may face challenges in accurately estimating gas usage for transactions involving state-dependent conditions, as it might not have complete or up-to-date information about the current state of the contract's storage during simulation.

Transaction complexity: The eth_estimateGas method might struggle to provide accurate gas estimates for complex transactions, particularly those involving multiple smart contract interactions, conditional execution paths, or nested function calls. These complexities can make it difficult to simulate the transaction execution and predict gas consumption accurately.

Variable-length loops: Similar to web3.eth.estimateGas, eth_estimateGas may not accurately predict the gas usage for functions with variable-length loops, as it might not have sufficient information about the number of iterations or the dynamic behavior of the loops during simulation.

Interactions with other smart contracts: The eth_estimateGas method might not fully account for the gas usage implications of interacting with external contracts, especially when those interactions involve varying implementations, state-dependent behavior, unpredictable gas consumption, or fallback functions.

Limiting factors:

Incomplete or outdated state information: The accuracy of eth_estimateGas can be impacted by the availability of complete and up-to-date information about the contract's storage during simulation. Incomplete or outdated state information can lead to inaccurate gas estimates.

Client-specific implementation: The implementation details of eth_estimateGas may vary between Ethereum clients, which can lead to slight differences in gas estimates. This inconsistency might cause challenges for developers in predicting accurate gas usage across different clients.

In conclusion, the eth_estimateGas JSON-RPC method is a widely-used approach for estimating gas usage in NFT transactions with dynamic arrays. However, it might not be the most accurate method for handling transactions involving state-dependent conditions, transaction complexity, variable-length loops, or interactions with other smart contracts. Developers may need to consider additional techniques, such as static analysis or adaptive gas estimation algorithms, to improve the accuracy of gas estimates in these scenarios.

3.3. eth_call JSON-RPC method with gas field:

The eth_call JSON-RPC method is primarily used for executing a read-only transaction against the Ethereum network without actually sending the transaction. It can also be utilized for gas estimation by including the gas field, which returns the gas used by the transaction during the simulation. In this section, we evaluate the eth_call method with the gas field for estimating gas usage in NFT transactions with dynamic arrays, discussing its effectiveness in providing accurate gas

estimates and identifying the challenges it faces in addressing state-dependent conditions, transaction complexity, and variable-length loops.

Effectiveness in providing accurate gas estimates:

Simulation-based estimation: Similar to the eth_estimateGas method, eth_call with the gas field simulates the transaction execution in a local environment to estimate gas consumption. This approach allows it to provide a reasonable approximation of the gas usage for various transaction types, including NFT transactions with dynamic arrays.

Challenges in addressing complexities:

State-dependent conditions: eth_call with the gas field might face challenges in accurately estimating gas usage for transactions involving state-dependent conditions, as it might not have complete or up-to-date information about the current state of the contract's storage during the simulation.

Transaction complexity: The eth_call method with the gas field might struggle to provide accurate gas estimates for complex transactions, particularly those involving multiple smart contract interactions, conditional execution paths, or nested function calls. These complexities can make it difficult to simulate the transaction execution and predict gas consumption accurately.

Variable-length loops: Similar to other simulation-based methods, eth_call with the gas field may not accurately predict the gas usage for functions with variable-length loops, as it might not have sufficient information about the number of iterations or the dynamic behavior of the loops during the simulation.

In conclusion, the eth_call JSON-RPC method with the gas field is another approach for estimating gas usage in NFT transactions with dynamic arrays. While it can provide reasonable gas estimates through transaction simulation, it might not be the most accurate method for handling transactions involving state-dependent conditions, transaction complexity, and variable-length loops. Developers may need to consider additional techniques, such as static analysis or adaptive gas estimation algorithms, to improve the accuracy of gas estimates in these scenarios.

3.4. Static analysis tools:

Static analysis tools, such as Remix and MythX, provide an alternative approach to gas estimation by analyzing the contract bytecode without executing the transaction. In this section, we examine the performance of these tools in estimating gas usage for NFT transactions with dynamic arrays, discussing their capabilities in bytecode analysis and how they fare in estimating gas usage for transactions involving state-dependent conditions, transaction complexity, and variable-length loops.

Capabilities in analyzing contract bytecode:

Bytecode analysis: Static analysis tools analyze the contract bytecode to identify patterns and estimate the potential gas consumption for each operation. This approach allows them to provide an approximation of gas usage without requiring a complete understanding of the contract's current state or simulating transaction execution.

Support for Solidity and other languages: Tools like Remix and MythX support analysis of contracts written in Solidity and other Ethereum-compatible languages, making them versatile options for developers working with various programming languages and contract types.

Performance in handling complexities:

State-dependent conditions: Unlike simulation-based methods, static analysis tools do not require complete information about the contract's storage state. However, they may still face challenges in accurately estimating gas usage for transactions involving state-dependent conditions, as their analysis is based on the contract bytecode, which may not provide sufficient information about the dynamic behavior of the contract.

Transaction complexity: While static analysis tools can identify complex patterns and operations in the contract bytecode, they might struggle to provide accurate gas estimates for transactions involving multiple smart contract interactions, conditional execution paths, or nested function calls. The static nature of bytecode analysis might not fully capture the dynamic aspects of complex transactions.

Variable-length loops: Static analysis tools may have difficulty estimating gas usage for functions with variable-length loops, as they might not have enough information about the number of iterations or the dynamic behavior of the loops based on the contract bytecode alone.

In conclusion, static analysis tools like Remix and MythX offer an alternative approach to gas estimation for NFT transactions with dynamic arrays by analyzing contract bytecode. While they provide some advantages over simulation-based methods, they might not be the most accurate method for handling transactions involving state-dependent conditions, transaction complexity, and variable-length loops. Developers may need to consider additional techniques, such as adaptive gas estimation algorithms or combining static analysis with dynamic analysis methods, to improve the accuracy of gas estimates in these scenarios.

3.5. Gas estimation services:

Gas estimation services, such as ETH Gas Station and Gas Now, provide users with recommended gas prices based on an analysis of recent network data. In this section, we assess the performance of these services in estimating gas usage for NFT transactions with dynamic arrays. We will discuss their approach to analyzing network data and providing gas price recommendations, as well as their limitations in estimating gas limits for specific transaction types.

Approach to analyzing network data and providing gas price recommendations:

Network data analysis: Gas estimation services collect and analyze data from recent transactions on the Ethereum network to determine trends in gas prices. They typically use statistical methods to identify the range of gas prices that resulted in successful transaction confirmations within a specific time frame.

Gas price recommendations: Based on their analysis, these services provide users with gas price recommendations to help them find a balance between transaction cost and confirmation time. The recommendations are often categorized into tiers, such as slow, standard, and fast, depending on the desired transaction confirmation speed.

Limitations in estimating gas limits for specific transaction types:

Focus on gas price: Gas estimation services primarily focus on providing gas price recommendations rather than estimating gas limits for specific transaction types. They might not have the necessary information or capabilities to analyze the gas usage of individual transactions, particularly those involving complex operations, such as NFT transactions with dynamic arrays.

Limited applicability to NFT transactions with dynamic arrays: As these services rely on aggregated network data, they might not accurately estimate gas usage for NFT transactions with dynamic arrays, which can have varying gas requirements depending on state-dependent conditions, transaction complexity, and variable-length loops.

Variability of network conditions: Gas estimation services base their recommendations on recent network data, which can be subject to rapid changes due to fluctuations in network congestion, gas prices, or other factors. As a result, their recommendations might not always be accurate or up-to-date, and users might need to monitor network conditions and adjust their gas prices accordingly.

In conclusion, gas estimation services like ETH Gas Station and Gas Now can provide useful gas price recommendations based on network data analysis but might not be the most suitable option for estimating gas limits for specific transaction types, such as NFT transactions with dynamic arrays. Developers and users may need to consider other gas estimation methods, such as web3.eth.estimateGas, eth_estimateGas JSON-RPC method, eth_call JSON-RPC method with gas field, or static analysis tools, to obtain accurate gas limit estimates for these transactions.

4.1. Description of the problem:

In this section, we present a case study focusing on out-of-gas errors in NFT transactions with dynamic arrays. We will describe the specific scenario where an array that tracks the number of NFTs a user has can cause the transaction to fail with an out-of-gas error due to inaccurate gas estimation.

Problem scenario:

Consider an NFT contract that uses a dynamic array to track the number of NFTs owned by each user. The contract includes functions to mint, transfer, and perform other operations involving NFTs. These functions may involve updating the dynamic array, which in turn can impact the gas usage of the transaction.

Causes of out-of-gas errors in this scenario:

State-dependent conditions: The gas usage of functions that interact with the dynamic array can vary depending on the current state of the contract's storage, such as the number of NFTs owned by the users or the length of the array. This state-dependent behavior can make it difficult for gas estimation algorithms to predict the correct gas limit, leading to underestimation and out-of-gas errors.

Transaction complexity: NFT transactions can involve multiple smart contract interactions, conditional execution paths, or nested function calls, which can further complicate gas estimation. The increased complexity can result in inaccurate gas estimates and contribute to out-of-gas errors.

Variable-length loops: Functions that iterate over the dynamic array might have a variable number of iterations based on the current state of the contract. The gas usage of these functions can vary significantly depending on the number of iterations, making it challenging for gas estimation algorithms to provide accurate estimates and increasing the likelihood of out-of-gas errors.

Interaction with other smart contracts: NFT transactions may involve interactions with external smart contracts, such as marketplaces or decentralized finance (DeFi) platforms. These interactions can introduce additional uncertainty in gas usage, further complicating the gas estimation process and increasing the chances of out-of-gas errors.

In summary, the case study highlights the challenges of accurately estimating gas usage in NFT transactions with dynamic arrays that track the number of NFTs owned by users. Factors such as state-dependent conditions, transaction complexity,

variable-length loops, and interactions with other smart contracts can contribute to inaccurate gas estimates, leading to out-of-gas errors and transaction failures.

4.2. Analysis of gas estimation methods:

In this section, we analyze how the popular gas estimation methods discussed earlier perform in the context of the case study involving NFT transactions with dynamic arrays. We will evaluate the accuracy of each method in estimating gas usage for the problematic NFT transactions and identify the factors contributing to their inadequacy in this specific scenario.

web3.eth.estimateGas: This method, based on transaction simulation, can struggle with accurately estimating gas usage for NFT transactions with dynamic arrays due to state-dependent conditions, variable-length loops, and interactions with other smart contracts. As the simulation relies on the current state of the contract, it may underestimate gas usage for transactions with varying gas requirements, leading to out-of-gas errors.

eth_estimateGas JSON-RPC method: Similar to web3.eth.estimateGas, the eth_estimateGas JSON-RPC method relies on transaction simulation and can face challenges in estimating gas usage for NFT transactions with dynamic arrays. State-dependent conditions, transaction complexity, and variable-length loops can contribute to inaccurate gas estimates and out-of-gas errors.

eth_call JSON-RPC method with gas field: While this method allows users to specify a gas limit for the transaction, it does not inherently estimate gas usage. Consequently, users need to make their own estimations or use other methods to determine an appropriate gas limit. This approach can still be prone to inaccuracies and out-of-gas errors in the context of NFT transactions with dynamic arrays.

Static analysis tools: Tools like Remix and MythX analyze contract bytecode to estimate gas usage, but may not accurately capture the dynamic behavior of NFT transactions with dynamic arrays. Their static analysis approach might not fully account for state-dependent conditions, transaction complexity, and variable-length loops, which can lead to inaccurate gas estimates and out-of-gas errors.

Gas estimation services: Services like ETH Gas Station and Gas Now focus primarily on providing gas price recommendations based on network data analysis, rather than estimating gas limits for specific transactions. As a result, they might not be the most suitable option for estimating gas usage in NFT transactions with dynamic arrays, leaving users to rely on other methods that may still result in inaccuracies and out-of-gas errors.

In conclusion, popular gas estimation methods face challenges in accurately estimating gas usage for NFT transactions with dynamic arrays. Factors such as state-dependent conditions, transaction complexity, variable-length loops, and interactions with other smart contracts can contribute to their inadequacy in this specific scenario, leading to out-of-gas errors and transaction failures.

4.3. Challenges faced by each method:

In this section, we will delve into the specific challenges each gas estimation method faces when dealing with NFT transactions involving dynamic arrays. We will discuss how state-dependent conditions, transaction complexity, variable-length loops, and interactions with other smart contracts can negatively impact the accuracy of these methods, leading to out-of-gas errors.

web3.eth.estimateGas and eth_estimateGas JSON-RPC method:

State-dependent conditions: These methods use transaction simulation, which is based on the current state of the contract. If the state changes during execution, the estimated gas usage might be inaccurate, leading to out-of-gas errors.

Transaction complexity: Complex transactions involving multiple smart contract interactions or conditional execution paths can introduce additional uncertainty in gas usage estimation, reducing the accuracy of these methods.

Variable-length loops: Functions with loops that have a variable number of iterations based on the current state of the contract can be challenging to estimate accurately, resulting in underestimated gas limits.

Interaction with other smart contracts: Gas usage might be influenced by the behavior of external contracts, which can introduce additional uncertainty and result in inaccurate gas estimates.

eth_call JSON-RPC method with gas field:

User-specified gas limit: This method does not inherently estimate gas usage but allows users to specify a gas limit for the transaction. Users need to make their own estimations or use other methods to determine an appropriate gas limit, which can still be prone to inaccuracies and out-of-gas errors.

Static analysis tools:

Limited dynamic behavior analysis: Static analysis tools like Remix and MythX might not fully capture the dynamic behavior of NFT transactions with dynamic arrays, as they analyze contract bytecode without considering the contract's current state.

Challenges with variable-length loops: These tools can struggle to accurately estimate gas usage for functions with variable-

length loops, as the number of iterations depends on the contract's state, which is not captured in the bytecode analysis.

External smart contract interactions: Static analysis tools might not account for gas usage influenced by external smart contracts, which can introduce additional uncertainty in the gas estimation.

Gas estimation services:

Focus on gas price: Services like ETH Gas Station and Gas Now primarily focus on providing gas price recommendations based on network data analysis, rather than estimating gas limits for specific transactions. This makes them less suitable for estimating gas usage in NFT transactions with dynamic arrays.

Limited applicability: As these services rely on aggregated network data, they might not accurately estimate gas usage for NFT transactions with dynamic arrays, which can have varying gas requirements depending on state-dependent conditions, transaction complexity, and variable-length loops.

In summary, the various gas estimation methods face specific challenges when dealing with NFT transactions involving dynamic arrays. Factors such as state-dependent conditions, transaction complexity, variable-length loops, and interactions with other smart contracts can negatively impact their accuracy, leading to out-of-gas errors and transaction failures.

5.1. Implementing safety margins:

In this section, we recommend implementing safety margins in gas estimation to account for inaccuracies and prevent out-of-gas errors. The rationale behind adding a safety margin is to ensure that transactions have sufficient gas to complete execution, even if the initial gas estimation is not entirely accurate. By providing a buffer, we can minimize the risk of out-of-gas errors, leading to a better user experience and improved network performance.

To implement safety margins, developers should consider the following guidelines:

Assess the complexity of the transaction: The complexity of the transaction and the contract's functions can give an indication of the potential variability in gas usage. More complex transactions or those with variable-length loops might require a larger safety margin than simpler transactions.

Analyze historical gas usage: Analyzing the historical gas usage of similar transactions can help developers determine a reasonable safety margin. By looking at past transactions, developers can identify the range of gas usage and set a safety margin accordingly.

Monitor gas estimation accuracy: Continuously monitoring the accuracy of gas estimation methods can help developers identify when safety margins need to be adjusted. If a particular method consistently underestimates gas usage, developers may need to increase the safety margin to prevent out-of-gas errors.

Strike a balance: Developers should aim to strike a balance between reducing the risk of out-of-gas errors and minimizing wasted gas. While a larger safety margin can help prevent out-of-gas errors, it might also result in wasted gas if the transaction does not consume the entire gas limit. Therefore, it is crucial to find an appropriate safety margin that accounts for potential inaccuracies without being overly conservative.

In conclusion, implementing safety margins in gas estimation can help mitigate the risk of out-of-gas errors in NFT transactions with dynamic arrays. By carefully considering the complexity of the transaction, analyzing historical gas usage, monitoring gas estimation accuracy, and striking a balance between error prevention and gas efficiency, developers can improve the reliability of their gas estimations and enhance the overall user experience.

5.2. Adaptive gas estimation algorithms:

In this section, we propose the development and adoption of adaptive gas estimation algorithms that can adjust to dynamic conditions in NFT transactions with dynamic arrays. These algorithms aim to improve gas estimation accuracy by considering factors such as state-dependent conditions, transaction complexity, variable-length loops, and interactions with other smart contracts.

The potential benefits of adaptive gas estimation algorithms include:

Improved accuracy: By continuously adapting to the changing conditions of the contract and the network, adaptive gas estimation algorithms can provide more accurate gas estimates, reducing the risk of out-of-gas errors and transaction failures.

Dynamic handling of state-dependent conditions: Adaptive gas estimation algorithms can account for state-dependent conditions by analyzing the current state of the contract and adjusting the gas estimation accordingly. This can lead to more precise gas estimates that consider the contract's current state and potential future states during execution.

Better handling of transaction complexity: Adaptive algorithms can take into account the complexity of transactions, including multiple smart contract interactions and conditional execution paths. By considering these factors, the algorithms can provide more accurate gas estimates that reflect the true gas requirements of complex transactions.

Variable-length loops consideration: Adaptive gas estimation algorithms can handle functions with variable-length loops

more effectively by estimating the number of iterations based on the current state of the contract and adjusting the gas estimate accordingly. This can help prevent underestimation of gas usage in functions with variable-length loops.

Interaction with other smart contracts: Adaptive algorithms can account for the behavior of external smart contracts during gas estimation. By considering the gas usage of interacting contracts, these algorithms can provide more accurate gas estimates that reflect the true gas requirements of transactions involving multiple smart contracts.

Learning from historical data: Adaptive gas estimation algorithms can leverage historical transaction data to improve the accuracy of future gas estimates. By analyzing past gas usage, these algorithms can learn patterns and trends that can inform better gas estimations for similar transactions in the future.

In conclusion, adaptive gas estimation algorithms offer significant potential in improving the accuracy of gas estimates for NFT transactions with dynamic arrays. By considering factors such as state-dependent conditions, transaction complexity, variable-length loops, and interactions with other smart contracts, these algorithms can lead to more accurate gas estimates, ultimately enhancing the overall user experience and network performance.

5.3. Testing and monitoring tools for gas estimation:

In this section, we emphasize the importance of testing and monitoring tools for gas estimation in the context of NFT transactions with dynamic arrays. These tools and techniques can help developers monitor gas usage, identify bottlenecks, and fine-tune gas estimation algorithms to improve their accuracy and reliability.

Smart contract testing frameworks: Testing frameworks, such as Truffle and Hardhat, allow developers to simulate transactions and analyze gas usage in a controlled environment. By running tests with various input scenarios and contract states, developers can identify potential issues with gas estimation algorithms and improve their accuracy.

Gas usage analyzers: Gas usage analyzers, such as Remix or Solc's built-in gas report, can provide insights into the gas consumption of specific functions within a smart contract. These tools can help developers identify areas where gas usage can be optimized and uncover potential bottlenecks that might cause out-of-gas errors.

Gas estimation services: Gas estimation services, like ETH Gas Station and Gas Now, can provide valuable data on the overall gas usage trends in the Ethereum network. By monitoring these trends, developers can adjust their gas estimation algorithms to account for changes in network conditions and improve the accuracy of their estimates.

Historical data analysis: Analyzing historical transaction data can help developers understand how gas usage has changed over time and identify trends that might affect their gas estimation algorithms. By incorporating this information into their algorithms, developers can create more accurate and adaptive gas estimation methods.

Monitoring and alerting tools: Real-time monitoring and alerting tools, such as Etherscan or Tenderly, can help developers keep track of gas usage in deployed contracts and detect potential issues with gas estimation in live environments. By receiving alerts when unexpected gas usage patterns are detected, developers can proactively address potential problems before they lead to out-of-gas errors.

Benchmarking and comparison: Comparing gas estimation results from different methods or services can help developers identify potential discrepancies and areas for improvement. By benchmarking their algorithms against other methods, developers can iteratively refine their gas estimation techniques to achieve better accuracy and reliability.

In conclusion, testing and monitoring tools for gas estimation play a crucial role in improving the accuracy and reliability of gas estimates for NFT transactions with dynamic arrays. By leveraging these tools and techniques, developers can optimize their gas estimation algorithms, identify and address potential issues, and ultimately enhance the overall user experience and network performance.

5.4. Best practices for NFT contract developers:

In this section, we provide a set of best practices for NFT contract developers to minimize the challenges associated with gas estimation in transactions involving dynamic arrays. By following these recommendations, developers can design and deploy more gas-efficient contracts, ultimately improving the overall user experience and network performance.

Design contracts with gas efficiency in mind: When developing NFT contracts, prioritize gas efficiency in the design and implementation of smart contract functions. Optimize storage usage, minimize unnecessary computation, and use efficient data structures to reduce gas costs.

Simplify complex transactions: Break down complex transactions into simpler, smaller transactions to make gas estimation more predictable and accurate. This can also improve the readability and maintainability of the smart contract code.

Avoid variable-length loops when possible: Minimize the use of variable-length loops, as they can make gas estimation challenging. Consider alternative approaches, such as pagination or limiting the number of iterations, to reduce uncertainty in gas usage.

Employ thorough testing and monitoring strategies: Use testing frameworks, such as Truffle and Hardhat, to simulate various transaction scenarios and analyze gas usage under different contract states. Implement monitoring and alerting

tools, such as Etherscan or Tenderly, to keep track of gas usage in deployed contracts and detect potential issues with gas estimation in live environments.

Benchmark and compare gas estimation methods: Compare the results of different gas estimation methods and services to identify potential discrepancies and areas for improvement. Continuously refine gas estimation techniques to achieve better accuracy and reliability.

Collaborate and learn from the community: Engage with the developer community to share experiences, learn from others' successes and failures, and adopt best practices in gas estimation and smart contract development.

Implement safety margins: Add a safety margin to gas estimates to account for inaccuracies and reduce the risk of out-of-gas errors. Choose an appropriate margin size that balances the need for reducing out-of-gas errors and minimizing wasted gas.

Stay up to date with Ethereum updates: Keep up with the latest developments in the Ethereum ecosystem, including protocol upgrades and new tools, to ensure your contracts remain efficient, secure, and up to date with best practices.

By following these best practices, NFT contract developers can minimize the challenges associated with gas estimation in transactions involving dynamic arrays and create more gas-efficient, reliable, and user-friendly NFT contracts.

In conclusion, accurately estimating gas usage for NFT transactions with dynamic arrays is a challenging task due to various factors such as state-dependent conditions, transaction complexity, variable-length loops, and interactions with other smart contracts. As we have seen, popular gas estimation methods like web3.eth.estimateGas, eth_estimateGas JSON-RPC method, eth_call JSON-RPC method with gas field, static analysis tools, and gas estimation services, all have limitations in addressing these challenges.

To overcome these difficulties, we have discussed recommendations and improvements such as implementing safety margins, developing adaptive gas estimation algorithms, using testing and monitoring tools, and following best practices for NFT contract developers. By adopting these strategies, developers can enhance the accuracy and reliability of gas estimation, ultimately improving the user experience and overall performance of the Ethereum network.

It is crucial for developers to stay up to date with the latest advancements in the Ethereum ecosystem and actively engage with the community to continuously refine their gas estimation techniques and smart contract development practices. As the Ethereum network evolves and new solutions are introduced, more efficient and accurate gas estimation methods will emerge, further improving the performance and scalability of decentralized applications involving NFTs and dynamic arrays.