Dynamic Gas Fee Adjustment System Reference Guide

System Overview

The dynamic gas fee adjustment system optimizes transaction costs while ensuring reliable transaction execution in varying network conditions. This guide explains the system's architecture, functionality, and configuration.

Core Components

Component	Location	Purpose
DynamicGasAdjuster	utils.ts	Adjusts gas parameters based on network conditions and transaction history
GasTransactionUtility	utils.ts	Wrapper that simplifies gas parameter management
GAS_OPTIMIZATION Constants	constants.ts	Configuration values for gas management
Implementation	smartContractService.ts	Integration with transaction processing

Key Constants

Constant	Value	Purpose
ESTIMATED_GAS_LIMIT	200000n	Default gas units for swap transactions
GAS_LIMIT_BUFFER	1.05	Standard safety margin (5%) for gas estimates
PRIORITY_FEE.LOW	2 Gwei	Base priority fee for low congestion
PRIORITY_FEE.MEDIUM	3 Gwei	Base priority fee for medium congestion
PRIORITY_FEE.HIGH	5 Gwei	Base priority fee for high congestion

Operational Workflow

1. Initialization

- System initializes with default multipliers (1.0)
- Fetches current network state
- Sets up regular network state monitoring

2. Network Analysis

- Fetches latest block data every 30 seconds
- Calculates block utilization (gasUsed/gasLimit)

- Determines network congestion level:
  - LOW: <50% utilization
  - MEDIUM: 50-80% utilization
  - HIGH: >80% utilization
- Sets appropriate base fee buffer

3. Gas Parameter Calculation

Copy

```
maxFeePerGas = baseFee + priorityFee + baseBuffer + dynamicBuffer
```

Where:

- **baseFee**: Current network base fee
- **priorityFee**: Congestion-based fee × priority multiplier
- **baseBuffer**: Congestion-based buffer (1-3 Gwei)
- **dynamicBuffer**: Additional buffer based on transaction history

Gas limit calculation:

Copy

```
gasLimit = ESTIMATED_GAS_LIMIT × congestionMultiplier
```

Where congestionMultiplier is:

- LOW: 1.05
- MEDIUM: 1.10
- HIGH: 1.20

4. Learning Mechanism

Success Response

- Records transaction details
- Resets consecutive failure counter
- Reduces fee multipliers by 5%
- Maintains minimum multiplier of 1.0

Failure Response

- Records failure type and details
- Adjusts parameters based on failure type:

Failure Type	Fee Multiplier Increase	Other Adjustments
Gas-related	+0.30	Adds 1 Gwei to buffer, +0.2 to priority multiplier
Stale quote	+0.10	None
Slippage	+0.05	None
Default	+0.05	None

- Applies exponential increases after 2 consecutive failures
- Caps multipliers to prevent excessive fees (max 3.0 for fee, 5.0 for priority)

Integration Example

typescript Copy

```typescript
// Obtain gas parameters
const params = await gasUtility.getGasParameters();

// Submit transaction with optimized parameters
const hash = await walletClient.writeContract({
  address: contractAddress,
  abi: contractAbi,
  functionName: 'executeArbitrage',
  args: [...],
  gas: params.gasLimit,
  maxFeePerGas: params.maxFeePerGas,
  maxPriorityFeePerGas: params.maxPriorityFeePerGas
});

// Process result and provide feedback to the adjuster
if (success) {
  gasUtility.recordSuccess(gasUsed, effectiveGasPrice, hash);
} else {
  gasUtility.recordFailure(errorType, gasPrice, hash);
}
```

Monitoring and Troubleshooting

Recommended Actions

The system provides recommendations based on transaction history:

- **PROCEED**: Normal operation (fee multiplier < 1.8, consecutive failures < 3)
- **CAUTION**: Potential issues (fee multiplier 1.8-2.5, consecutive failures 3-4)
- **PAUSE**: Critical issues (fee multiplier > 2.5, consecutive failures ≥ 5)

Diagnostic Information

Call `gasUtility.getAdjustmentStats()` to retrieve:

- Current fee and priority fee multipliers
- Base fee buffer
- Consecutive failures count
- Network congestion level
- Recent transaction statistics

Benefits

1. **Cost Optimization**: Lower fees during favorable network conditions
2. **Reliability**: Ensures transaction success during high congestion
3. **Adaptability**: Responds to changing network conditions
4. **Self-Improvement**: Learns from past transactions to improve success rates
5. **Arbitrage Efficiency**: Maximizes capture of profitable trading opportunities

This system helps ensure that profitable arbitrage opportunities aren't missed due to gas-related transaction failures, while also avoiding unnecessary overpayment for gas.