(Alpha) Priority Fee API

This endpoint is in active development.

What Are Priority Fees on Solana?

On Solana, you can add a fee to your transactions to ensure they get prioritized by validators. This is especially useful if i) the current block is near-full, and ii) the state you're trying to write to is highly contested (for example: a popular NFT mint).

How Do You Know What Fees to Use?

For priority fees to make sense — you need to compute them properly. This is trickier than it sounds due to ever-changing network conditions.

Existing Solution

Currently, the Solana RPC spec supports a method called getRecentPriorityFees . This method pulls data from a cache on an RPC node that stores up to the last 150 blocks. The response is defined by GetRecentPriorityFeesResponse: type

GetRecentPriorityFeesResponse

=

Vec < RpcPrioritizationFee

struct

RpcPrioritizationFee

{ slot :

u64 prioritization_fee :

u64 } Here, prioritization_fee is calculated on a per-slot basis: prioritization_fee(slot) = MAX(min_txn_fee, MAX(min_writeable_account_fee(tx)) This provides an idea of the minimum value to set for fees, so is very limited in its usefulness. Additionally, it provides a list of values, which puts more burden on developers to figure out how to use it. See the code here (from Solana 1.16.21).

Helius Priority Fee API — An Improved Solution

We define a new method, getPriorityFeeEstimate , that simplifies the response into a single value. Most importantly: it considers both global and local fee markets. The method uses a set of predefined priority levels (percentiles) to dictate the returned estimate. Users can optionally specify to receive all the priority levels and adjust the window with which these are calculated via lookbackSlots fn

get_recent_priority_fee_estimate (request :

GetPriorityFeeEstimateRequest)

->

GetPriorityFeeEstimateResponse struct

GetPriorityFeeEstimateRequest

{ transaction :

Option < String

,

// estimate fee for a serialized txn account_keys :

Option < Vec < String

,

// estimate fee for a list of accounts options :

Option < GetPriorityFeeEstimateOptions

} struct

GetPriorityFeeEstimateOptions

{ priority_level :

Option < PriorityLevel

,

// Default to MEDIUM include_all_priority_fee_levels :

Option < bool

,

// Include all priority level estimates in the response transaction_encoding :

Option < UiTransactionEncoding

,

// Default Base58 lookback_slots :

Option < u8

,

// number of slots to look back to calculate estimate. Valid number are 1-150, defualt is 150 } enum

PriorityLevel

{ NONE ,

// 0th percentile LOW ,

// 25th percentile MEDIUM ,

// 50th percentile HIGH ,

// 75th percentile VERY_HIGH ,

// 95th percentile // labelled unsafe to prevent people using and draining their funds by accident UNSAFE_MAX ,

// 100th percentile DEFAULT ,

// 50th percentile } struct

GetPriorityFeeEstimateResponse

{ priority_fee_estimate :

Option < MicroLamportPriorityFee

priority_fee_levels :

Option < MicroLamportPriorityFeeLevels

} type

MicroLamportPriorityFee

=

f64 struct

MicroLamportPriorityFeeLevels

{ none :

f64 , low :

f64 , medium :

f64 , high :

f64 , very_high :

f64 , unsafe_max :

f64 , }

Examples

Request all priority fee levels for Jup v6 { "jsonrpc" :

"2.0" , "id" :

"1" , "method" :

"getPriorityFeeEstimate" , "params" :

[{ "accountKeys" :

["JUP6LkbZbjS1jKKwapdHNy74zcZ3tLUZoi5QNyVTaV4"], "options" :

{ "includeAllPriorityFeeLevels" :

true } }] } Response { "jsonrpc" :

"2.0" , "result" :

{ "priorityFeeLevels" :

{ "min" :

0.0 , "low" :

2.0 , "medium" :

10082.0 , "high" :

100000.0 , "veryHigh" :

1000000.0 , "unsafeMax" :

50000000.0 } }, "id" :

"1" } Request the high priority level for Jup v6 { "jsonrpc" :

"2.0" , "id" :

"1" , "method" :

"getPriorityFeeEstimate" , "params" :

[{ "accountKeys" :

["JUP6LkbZbjS1jKKwapdHNy74zcZ3tLUZoi5QNyVTaV4"], "options" :

{ "priority_level" :

"HIGH" } }] } Response { "jsonrpc" :

"2.0" , "result" :

{ "priorityFeeEstimate" :

1200.0 }, "id" :

"1" }

Sending a transaction with the Priority Fee API (Javascript)

This code snippet showcases how one can transfer SOL from one account to another. In this code, the transaction is passed to the priority fee API which then determines the specified priority fee from all the accounts involved in the transaction. const

{ Connection , SystemProgram , Transaction , sendAndConfirmTransaction , Keypair , ComputeBudgetProgram , }

=

require ("@solana/web3.js"); const bs58 =

require ("bs58"); const HeliusURL =

"https://mainnet.helius-rpc.com/?api-key=" ; const connection =

new

Connection (HeliusURL); const fromKeypair = Keypair . fromSecretKey (Uint8Array . from ("[Your secret key]"));

// Replace with your own private key const toPubkey =

"CckxW6C1CjsxYcXSiDbk7NYfPLhfqAm3kSB5LEZunnSE" ;

// Replace with the public key that you want to send SOL to async

function

getPriorityFeeEstimate (priorityLevel , transaction)

{ const response =

await

fetch (HeliusURL ,

{ method :

"POST" , headers :

{

"Content-Type" :

"application/json"

}, body :

JSON . stringify ({ jsonrpc :

"2.0" , id :

"1" , method :

"getPriorityFeeEstimate" , params :

[{ transaction : bs58 . encode (transaction . serialize ()),

// Pass the serialized transaction in Base58 options :

{

priorityLevel : priorityLevel }, },], }), }); const data =

await response . json (); console . log ("Fee in function for" , priorityLevel , " :" , data . result . priorityFeeEstimate); return data . result ; } async

function

sendTransactionWithPriorityFee (priorityLevel)

{ const transaction =

new

Transaction (); const transferIx = SystemProgram . transfer ({ fromPubkey : fromKeypair . publicKey , toPubkey , lamports :

100 , }); transaction . add (transferIx); transaction . recentBlockhash =

(await connection . getLatestBlockhash ()). blockhash ; transaction . sign (fromKeypair); let feeEstimate =

{

priorityFeeEstimate :

0

}; if

(priorityLevel !==

"NONE")

{ feeEstimate =

await

getPriorityFeeEstimate (priorityLevel , transaction); const computePriceIx = ComputeBudgetProgram . setComputeUnitPrice ({ microLamports : feeEstimate . priorityFeeEstimate , }); transaction . add (computePriceIx); } try

{ const txid =

await

sendAndConfirmTransaction (connection , transaction ,

[fromKeypair ,]); console . log (Transaction sent successfully with signature { txid }); }

catch

(e)

{ console . error (Failed to send transaction: { e }); } } sendTransactionWithPriorityFee ("High");

// Choose between "Min", "Low", "Medium", "High", "VeryHigh", "UnsafeMax"

(Appendix) Calculating the Percentiles

To calculate the percentiles we need to consider the global and local fee market over transactions in the last N slots For example, priority_estimate(p: Percentile, accounts: Accounts) = max(percentile(txn_fees, p), percentile(account_fees(accounts), p)) where txn_fees are the txn_fees from the last 150 blocks, and account_fees(accounts) are the fees for txns containing these accounts from the last 150 blocks. Here we are considering the total set of fees seen for accounts and transactions, as opposed to the minimum.

Global Fee Market Estimate

The global fee market estimate is a percentile of priority fees paid for transactions in the last N slots.

Local Fee Market Estimate

The local fee market is influenced by the number of people trying to obtain a lock on an account. We can estimate this similarly to the global fee market, but instead use the percentile of fees paid for transactions involving a given account(s). If a user requests an estimate for multiple accounts in the same transaction, we will take the max of the percentiles across those accounts.

Priority Fee Estimate

The priority_fee_estimate will be the max of the global and local fee market estimates.

Extensions

This method could also be integrated into simulateTransaction and returned with the response context. This way developers using simulateTransaction can eliminate an extra RPC call Solana RPC Nodes -Previous RPC Proxy — Stop Your API Keys from Leaking Next- Compression & DAS API What is Compression on Solana?Last modified1mo ago