AIP[1]-015: Gas Fee Charging

platfowner, liayoo 2021-03-06

Goals

- Introduce Gas Fee Charging Protocol to AIN Blockchain

Problem Definition

We cover the following topics:

- Define the basic units of gas fee
- Determine how to select transactions in each block (see Ethereum Gas & Fee for reference)
- Determine how to charge the gas fee of the selected transactions
- Determine how to make the gas fee paid

Requirements

The gas fee charging protocol has the following requirements:

- Should cover the most critical resources so as to help the system stability by containing critical resource abuses
- Should be fair and reasonable from the token economy perspective
- The simpler the better

Terminology

We use the following definitions:

- **Gas amount**: The amount of gas units
- **Gas price**: The price of each gas unit
- **Gas cost**: (gas amount) * (gas price)

[1] AI Network Improvement Proposal. Visit https://docs.ainetwork.ai for the full list.

Proposed Design

Key Ideas

We classify transactions into two categories depending on the paths of their operations:

- **Service transactions**: Transactions for paths **not** under /apps, e.g. /transfer
- **App transactions**: Transactions for paths under /apps, e.g. /apps/afan

There are two types of gas fees ((i.e., the most critical resources of AIN Blockchain):

- **Bandwidth gas fee**:
    - Controls the transaction throughput
    - The number of write operations and external RPC calls done by the transactions
- **State gas fee**:
    - Controls the size of states
    - Service transaction: The number of accounts (individual or service) created by the transaction
    - App transaction: The state tree size of the app in the state database

We support two types of **charging methods**:

- For service transactions, we apply **by-transfer charging** for each transaction where gas fees are paid by money transfer. This is applied to transactions individually by specifying gas price in each transaction.
- For app transactions, we apply **by-staking charging** for gas fee budgets per app where gas fees are paid by money staking (no transfer). This is applied to transactions in groups per app.

When transactions compete each other we apply the following **selection criteria**:

- Among service transactions, we select high gas price transactions first
- Among app transactions, we apply separate gas fee budgets per app

Design Details

Gas Fee Budgets

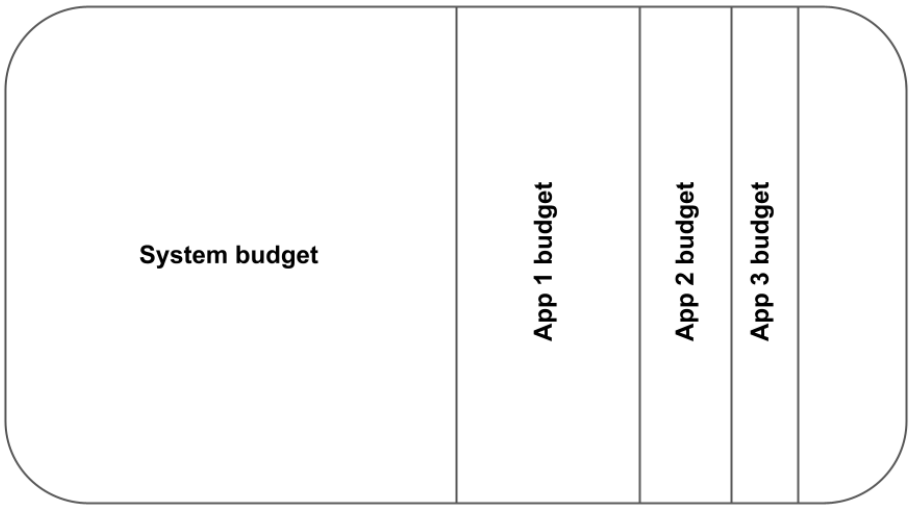

Figure 1. Gas budgets.

From the most critical resources (see Critical Resources), we define two types of gas budgets:

- **Bandwidth budget:** Maximum number of **write operations** and **external RPC calls** allowed in a block
    - In SET operation, the size of the op_list array
    - In native function triggering, the number of all write operations executed by the triggered functions (including **chained** function calls)
    - In non-native function triggering, the number of external RPC calls performed by the triggered functions (including **chained** function calls)
- **State budget:** The maximum state tree size allowed in the state database

The bandwidth budget is divided into two:

- **Service bandwidth budget** for all service transactions
- **App bandwidth budget** for all app transactions

The service bandwidth budget cannot exceed 50% of the total bandwidth budget. The app bandwidth budget is again divided into per-app budgets, which is proportional to the per-app staking amount. By dividing the budgets into app-level, app operators can run their apps in a stable manner. When the total use of the service bandwidth budget is less than 50%, it can be reallocated for app transactions. So, the actual total app bandwidth budget is **(total bandwidth budget) - (total service bandwidth gas fee in use)**. The over-bandwidth-budget transactions remain in the transaction pool waiting for the next chances.

The state budget is also divided into two:

- **Service state budget** for all service transactions
- **App state budget** for all app transactions

The service state budget and the app state budget cannot exceed 50% of the total state budget, respectively. In addition to this category-level service state budget, we apply **account registration fee** for state use:

- When new accounts (individual or service) are registered by the transaction, extra gas amount is added to the bandwidth gas amount depending on how many accounts are registered (see Charging Methods for more details)

Similarly to the app bandwidth budget, the app state budget is again divided into per-app budgets, which is proportional to the per-app staking amount. Again, this app-level state budget helps app operators run their apps in a stable manner. The over-state-budget app transactions are rejected to be added to the transaction pool. Notably we have a plan to add auto state optimization features for states.

Table 1. Gas budgets.

Gas budget	Transaction category	Budget application level	Budget cap	When applied	Over-budget handling	Notes
Bandwidth budget	Service transaction	Per transaction category	50% of the total bandwidth budget	Tx pool -> block	Not selected in block	
	App transaction	Per app	Prop. to the per-app staking amount among **unused** by service transactions	Tx pool -> block	Not selected in block	
State budget	Service transaction	Per transaction category	50% of the total state budget	RPC API -> Tx pool	Rejected to be added to tx pool	
	App transaction	Per app	50% of the total state budget	RPC API -> Tx pool	Rejected to be added to tx pool	

Table 1 shows a summary of the gas budgets.

Charging Methods

In addition to the budgets (i.e., by-staking charging method), we apply a **by-transfer charging** method which is done by actual account-to-account money transfer. This is applied only to service transactions and its payment is done at the transaction level. Table 2 shows a summary of the application of charging methods.

Table 2. Application of charging methods.

Transaction category	Gas fee type	Charging method	Charging level	Notes
Service transaction	Bandwidth gas fee	By transfer	Per transaction	
	State gas fee	By transfer	Per transaction	
App transaction	Bandwidth gas fee	By staking	Per app	
	State gas fee	By staking	Per app	

For the by-transfer charge, the transaction sender specifies a **gas price** to be paid for the transaction's gas amount when it's accepted. The transaction's gas amount is the sum of the bandwidth gas amount and the state gas amount. The state gas amount is added only for service transactions and it has positive values only when a new (individual and service) account is registered in the state database (i.e., /accounts/0x_an_address or /service_accounts/<service type>/<service name> is created) by the transaction. So, it's called **account registration fee**.

In addition to the account registration fee, we have the following ideas for state optimization:

- Auto-deletion of stale states (e.g. transfer results)
- Archiving unused accounts

Transaction Selection Policy

As mentioned above, the over-state-budget transactions are rejected by the RPC API even before the transaction pool stage. So they don't have any chance to be selected in blocks. The over-bandwidth-budget transactions, however, are handled on the block proposal stage by not being selected in blocks. In this case we need to decide which transactions to be added first. Here is the rules applied:

- Among service transactions, those of higher gas prices have priorities
- Among app transactions, those of earlier timestamps have priorities

Since an AIN blockchain node executes transactions as soon as they arrive, we can record the initial execution time by adding an `**executedAt**` field to the transactions' extra data. This is when the state budget check occurs as well. At this stage, *unordered* transactions will have no trouble with nonces or timestamps, but *ordered* transactions will get dropped if their timestamps are not higher than the latest `/accounts/${address}/timestamp` value. *Numbered* transactions with higher nonces than `/accounts/${address}/nonce` value will fail to be executed, but will not get dropped yet. Instead, they are added to a 'numbered tx pool' and later tried again when transactions with earlier nonces are executed. Transactions that are successfully executed and

added to the transaction pool will also have `**bandwidthGasAmount**` recorded in their extra data.

Transactions from the same address are first sorted in the order of their `exectedAt` timestamps and nonces. Then the block proposer needs to select and apply some ordering to transactions from multiple addresses to collect and include in a block. There are three cases to consider when transactions are signed by different addresses:

1. Service transaction vs. service transaction
2. App transaction vs. service transaction
3. App transaction vs. app transaction

If both are service transactions (case 1), we prioritize the one with a **higher gas price**. In the second case, **service** transactions have priorities, since they pay gas. If both are app transactions (case 3), one with a **lower `executedAt` value** is included first.

Table 3. A summary of the transaction selection policy.

vs.	Service transaction	App transaction	Notes
Service transaction	Higher gas price first	Service transaction	
App transaction	Service transaction	Earlier timestamp first	

Table 3 shows a summary of the transaction selection policy.

After merging the sorted per-address transaction lists, according to the criteria mentioned above, the service and app gas budget checks are done to the merged list. Traversing through the merged list, only the transactions that do not exceed the service / app gas budget limits will be accepted into the final list, which will be the transaction list of a new block.

Gas Fee Collection

As mentioned above, the bandwidth gas fee for the service transactions are paid by account-to-account money transfer and it's done for each transaction. To make this process clear and automatic, we support the following features:

- We use a special account for the gas fee collection on **/service_accounts/gas_fee/gas_fee/<block number>/balance** e.g. /service_accounts/gas_fee/gas_fee/10000/balance. We call this **gas fee account**.
- For the gas fee calculation and transfer, native function **_collectFee()** is automatically invoked in each transaction execution. It's done by a SET_VALUE operation on **/gas_fee/collect/<from account>/<block number>/<tx_hash>/**{
  amount: <amount to collect>
  }

e.g. / gas_fee / collect / 0x_A_User / 10000 / 0x_a_tx_hash {
amount: <amount to collect>
}

- If a transaction is associated with a Billing Account, the gas fee is paid by the associated billing account. Otherwise it's paid by the individual account (i.e., by the key signed account).

The gas fee for app transactions is paid by per-app staking (i.e., by-staking charging method). To change gas fee budgets for an app, the owner of the app can stake or un-stake money. It can be done by triggering native functions:

- **_stake()**: Stake more money for an app
  - It's triggered by a SET_VALUE operation on **/staking/<app name>/stake/<key>/**{
    amount: <staking amount>
    }
- **_unstake()**: Unstake some portion from the staked money for an app
  - It's triggered by a SET_VALUE operation on **/staking/<app name>/unstake/<key>/**{
    amount: <unstaking amount>
    }
  - This can be called only by the admins of the app
  - The portion of the staked money allowed for unstaking is limited by the spare tree size of the app in the state database.

The staked money will be saved in a service account: **/service_accounts/staking/<app name>** (e.g. /service_accounts/staking/afan). The balance of this account is used for the gas budget calculation of the app.

For the redistribution of the collected gas fee, see Gas Fee Redistribution.

Gas Price In Transaction

Table 4. Structure of a tx body of a service transaction.

```
{
  operation: { type, ref, value } | { type: 'SET', op_list },
  nonce: <number>,
  timestamp: <number>,
  gas_price: <number>
}
```

Table 4 shows how a gas price can be set for a service transaction. The total gas fee paid by the user / billing account for the particular transaction would be **gas price * number of gas units**.

Billing Accounts

For service transactions, gas fees are deducted from individual or billing accounts, depending on the service type's app dependency. For app-dependent services, such as payments and staking, the app's billing account is used (`/service_accounts/billing/<app_name>`). On the other hand, for app-independent services, such as transfer, escrow, and sharding, the individual account of the transaction signer is used.

For app transactions, initially, we're planning not to charge any gas fees. In the future, as apps get bigger, we might charge them gas fees as well. In that case, the app's billing account (`/service_accounts/billing/<app_name>`) will be used. For more information regarding billing accounts, see Billing Accounts.

Further Extensions

We have the following ideas for further extensions:

- Allow adaptation of the gas fee parameters by the block proposers, e.g. x% increase / decrease of the total gas budgets depending on the machine performance upgrades
- Use average gas price in the transaction selection instead of individual gas prices where average_gas_price = gas_cost_sum / gas_amount sum.

Conclusion

We proposed a design for the gas fee charging. It can be summarized as follows:

- It's based on the most critical resources: **bandwidth** and **state tree size**.
- It adopts two charging methods: **by-transfer** and **by-staking**
- It supports application developers by allowing:
    - Budgets by staking
    - App-level charging
    - Use of the unused portion of the service gas budgets

Links

- Ethereum Gas & Fees (link)
- AIP-014: Critical Resources of Blockchain Service (link)
- AIP-017: Billing Accounts (link)

- AIP-011: Service Account & Transfer (link)
- AIP-016: Gas Fee Redistribution (link)

Document History

Date	Who	Change	Notes
2021-03-06	platfowner	Initial draft	
2021-03-08	platfowner, liayoo	Internal review: - Gas Fee unit == # of write operations	
2021-03-15	minsulee2, liayoo, cshcomcom, platfowner	High-level internal review	
2021-03-18	platfowner	Major revision	
2021-03-22	liayoo, platfowner	Detailed internal review & revision: - Transaction execution order - _openApp() - Gas price in transaction	
2021-03-24	platfowner, liayoo	Transaction selection policy in block creation	
2021-03-25	platfowner	Account registration fee	
2021-04-16	platfowner	External RPC calls	
2021-05-12	platfowner	Github IDs Link to full list	
2021-05-12	platfowner	Published	