Smart contracts cannot subscribe to events, and therefore all the actions need to be triggered by an entity external to the contract.

I have been working on a system that provides a solution to this (I am aware of Ethereum Alarm Clock). Previously I posted a system and named Chronos that allowed a contract to be called at some time or block using a callback function. This has the issue of not being general enough as being able to schedule transaction with arbitrary bytecodes is more desirable (and more useful).

The primary idea in scheduling transactions is that users can build transactions and submit them to an executor that will broadcast the transaction at the time determined by the subject. As it is necessary that the executor doesn’t submit the transaction previous to the time or block indicated by the user, a smart contract can be implemented to control this.

I implemented a system that allows to schedule transactions with arbitrary bytecode in a trustless fashion. This is how it works:

A user (smart contract or EOA) send information to the scheduling system smart contract with the following information:

Parameter

Description

blocknumber (uint256)

Block or timestamp at which the transaction should be executed (see schedType parameter)

to (address)

recipient of the transaction

value (uint256)

Amount of Wei to transfer

gaslimit (uint256)

Maximum gas to spend on the transaction

gasprice (uint256)

gas price to be used in the transaction

data (bytes)

transaction data

schedType (bool)

true: schedule using a timestamp, false: schedule using blocknumber

The system creates an account for the user if it doesn’t exist already, (I would show later that this produces significant savings in gas). An id is generated for the request, the data is logged, and the only the hash of the parameters submitted is saved. This data is indexed with the id previously generated for this particular request.

The user should provide the cost of the transaction (gas) plus a fee. This is stored in the user accounts and can be retrieved by the system only if the request is executed. Any ether value that is part of the transaction is also saved in the user account.

When the time of execution arrives, the executor calls the contract (calling before the correct time will revert making the executor to lose gas, and not getting the fee, enforcing the execution at the right time). It verifies the call data using the hash of the user parameters (stored in the contract) and sends the data to the user account which has the necessary function to execute the transaction. The gas left is returned to the user that generated the request, and the fee is charged.

If the call requested by the subject fails (out of gas, revert, etc.) the contract catch the exception and return the remaining gas and any ether that is part of the transaction to the original requester. Data of transactions successfully executed is deleted, keeping controlled the amount of storage associated with the contract.

Furthermore, the scheduled transaction can be canceled by the user that created it without incurring in any payment to the system.

Regarding gas, the user account creation cost is paid only once, after the first transaction the scheduling of any transactions is as low (depending on the amount of data) as 70000 gas. Executing the transfer of ether cost about 55000 gas. The cost of creating an account is about 250000 gas which as I describe above is done only once.

Concerning the fee, it can be updated to compensate for changes in the value of Ether/USD, but here also a mechanism to avoid abuse has been implemented. This mechanism does not

allow changing the fee by more than 10%, and it can be changed only once per day. The current fee in the system is 0.0005ETH ~10cents and is meant to stay like that.

You can see the smart contract, description and exampleshere

Also for EOAs, I developed a web app that interface with the smart contract through Metamaskhere

The system is online on the mainnet and ropsten:

Network

Address

MainNet

0xCBe7AB529A147149b1CF982C3a169f728bC0C3CA

Ropsten

0xFcFB45679539667f7ed55FA59A15c8Cad73d9a4E

I am aware of the Ethereum Alarm Clock. It was down for a long time, and now the team from Chronologic has rebuilt it. Their system has "time nodes" (executors) that need to hold a minimum amount of a token from Chronologic. Also the cost of using the system is very high, the amount of gas to schedule a transaction is above 500000 gas and a bounty should be offered to the time nodes to pick up the transaction (furthermore a fee to the -original developer of the EAC seems to be charged as well, this is not clear) . This features of the system based on EAC motivated me to develop the system that I am proposing (I called it Aion after the Hellenistic deity associated with time)

I am looking forward to hear your comments.