Smart Contract Signals and Slots Formalization

May 2020

1 Introduction

The aim of this document is to provide a theoretical framework for how signals and slots might be implemented on top of a real blockchain network. In our case, we chose to implement the feature on top of Ethereum. As a result, a familiarity with the Ethereum Yellow Paper is expected. In a separate document we will build a simple theoretical virtual machine from scratch to better demonstrate how such a feature looks.

The following are a few important values and functions used throughout the document.

- σ: world state
- μ: machine state
- α: account address
- `KEC()`: Keccak-256 hash function
- `RLP()`: Recursive length prefix serialization
- `TRIE()`: Returns the root value of the trie

2 Signals and Slots

We denote a signal instance, E, as a tuple with the following fields:

- owner, E_a: The address of the contract that owns this signal.
- identifier, E_i: A unique identifier associated with a signal that is defined during contract creation. This is accomplished by assigning each signal a unique ephemeral $sigLocalId$ during contract creation. $\texttt{KEC}(E_a + sigLocalId)$ can then be used to generate a unique identifier.
- data, E_d: An infinite size byte array containing the output data of the signal.

We denote a slot instance, L, as a tuple with the following fields:

- owner, L_a: The address of the contract that owns this slot. This is also the address that pays for the gas consumed during the execution of the slot.
- code, L_c: A pointer to executable EVM code that is the entry point to the slot.
- gasPriceRatio, L_p: A scalar value greater or equal to zero indicating how much is willing to be payed for gas relative to the average gas price of regular transactions. The higher this value, the more likely this slot transaction will be mined.
- gasLimit, L_g: A scalar value equal to the maximum amount of gas that should be used in executing this slot.

Note: each slot can be attached to only one signal while each signal can be listened by multiple slots.

3 World State

All entities have their necessary information stored in an account, represented by a 20-byte address α. The world state is a mapping between addresses and account states. An account state $\sigma[\alpha]$ has the following five fields:

- nonce, $\sigma[\alpha]_n$: A scalar counter recording the number of previous activities initialized by this account.
- balance, $\sigma[\alpha]_b$: A scalar value representing the number of Wei owned by this account.
- storageRoot, $\sigma[\alpha]_s$: Hash of the root node of the trie that encodes the storage content of this account.
- codeHash, $\sigma[\alpha]_c$: Hash of the EVM code that gets executed when $\sigma[\alpha]$ receives a message call. This is immutable once established.
- slotRoot, $\sigma[\alpha]_l$: Hash of the root node of the trie that maps E_i to $\texttt{RLP}(\texttt{LIST}(L))$.
- slotQueue, $\sigma[\alpha]_{lq}$: Hash of the queue that contains all slot transactions related to this account.

Therefore an account state $\sigma[\alpha]$ can be represented as the following tuple:

$$\sigma[\alpha] \equiv (\sigma[\alpha]_n, \sigma[\alpha]_b, \sigma[\alpha]_s, \sigma[\alpha]_c, \sigma[\alpha]_l, \sigma[\alpha]_{lq})$$

4 Slot Transaction

A slot Transaction ST is a transaction generated as a response to a signal emitted from $\sigma[\alpha_{emitter}]$. These transactions do not require any signature as its validation is done according to the slot trie and queue maintained in the world state as well as block headers. The only two fields included in a ST are:

- signal, ST_e: A signal tuple instance E of interest.
- slot, ST_l: A slot tuple instance L of interest.

5 Block Header

The block header H is a collection of relevant pieces of information:

- parentHash, H_p: Keccak 256-bit hash of the parent block's header.
- refereeHash, H_o: serialized RLP sequence of the referee list consisting of Keccak 256-bit hashes of referee blocks.
- author, H_a: address of the author.
- transactionRoot, H_t: Keccak 256-bit hash of the root node of transaction trie.
- deferredStateRoot, H_r: Keccak 256-bit hash of the root node of the state trie after "stable transactions" are executed.
- deferredReceiptsRoot, H_e: Keccak 256-bit hash of the root node of the receipt trie during the construction of deferredStateRoot.
- deferredLogsBloom, H_b: bloom filter for logs of transactions receipts included.
- blame, H_m: A scalar value for evaluating ancestor blocks.
- difficulty, H_d: Value corresponding to the difficulty of the block.
- number, H_i: A scalar value equal to the number of ancestor blocks.
- adaptiveWeight, H_w:
- height, H_h: number of parent references to reach the genesis block.
- gasLimit, H_l: scalar value to the current limit of gas expenditure per block.
- timestamps, H_s: Unix time.
- nonce, H_n: Value that proves that a sufficient amount of work has been carried out on this block.
- slotTransactionRoot, H_{st}: Hash of the root node of slot handler trie. The trie maps $\texttt{KEC}(blockNumber)$ to $\texttt{RLP}(\texttt{QUEUE}(ST))$. This trie holds the list of transactions that need to be queued at each later block number. This trie is periodically pruned as blocks get mined.

Currently, every block B consists of two parts: a block header H and a list of transactions Ts. On top of this block structure, we are adding list of special transactions, STs. Therefore the block B can be represented as follows:

$$B \equiv (B_H, B_{Ts}, B_{STs})$$

Upon the mining of a block, the following state change is made to keep the slot queues in each account up to date. Let B be the previous block header and B' be the new block header.

$$\begin{aligned} &\textit{while } B_{st}[B'_i] \neq \varnothing : \\ &\quad ST = B_{st}[B'_i].\texttt{DEQUEUE}() \\ &\quad \sigma[\{ST_l\}_a]_l q.\texttt{ENQUEUE}(ST) \end{aligned}$$

6 Execution Environment

The list of opcodes we need for implementing the proposed event-driven smart contract design. Borrowing the notation from the Ethereum Yellow Paper, we assume O is the EVM state-progression function and define the terms pertaining to the next cycle's state (σ, μ) such that:

$$O(\sigma, \mu, A, I) \equiv (\sigma', \mu', A', I)$$

where σ represents the active memory or the system state, μ is the storage used, A is the accrued substate (information acted upon immediately following the transaction), and I is some pieces information used in the execution environment.

The list of information is as listed below

Variable	**Description**
A_s	the self-destruct set: a set of accounts that will be discarded following the transaction's completion
A_l	log series
A_t	touched accounts
A_r	the refund balance
$\mu_{\mathbf{s}}$	machine's stack
$\mu_{\mathbf{m}}$	machine's memory
μ_i	the active number of words in memory (counting continuously from position 0)
μ_g	gas available
μ_{pc}	the program counter
I_a	the address of the account which owns the code that is executing
I_o	the sender address of the transaction that originated this execution
I_p	the price of gas in the transaction that originated this execution
I_d	the byte array that is the input data to this execution; if the execution agent is a transaction, this would be the transaction data
I_s	the address of the account which caused the code to be executing
I_v	the value, in Wei, passed to this account as part of the same procedure as execution
I_b	the byte array that is the machine code to be executed.
I_H	the block header of the present block
I_e	the depth of the present message-call or contract-creation (i.e. the number of CALLs or CREATEs being executed at present)
I_w	the permission to make modifications to the state

7 Slot Transaction Execution

This section formalizes how slot transactions are executed. Firstly a slot transaction is popped off H_{st} and the addresses listener changes state. Next, a gas price and limit are determined and an upfront cost is charged to the slot account. Finally, an execution environment is set up and executed in the same way as a regular transaction.

State Change: To get an ST from account α, the following state change takes place:

$$ST = \sigma[\alpha]_{lq}.\texttt{DEQUEUE}()$$

Gas Price: To execute a slot transaction, we need to determine the gas price I_p. We calculate this by multiplying the average gas price of regular transactions in the previous block by $\{ST_l\}_p$. Let the previous block be denoted as B', hence the transactions in the previous block is B'_{Ts}.

$$I_p = \{ST_l\}_p \cdot \frac{\sum_{T \in B'_{Ts}} T_p}{|B'_{Ts}|}$$

Gas Limit: The gas limit is set to $\{ST_l\}_g$.

Intrinsic Gas: Intrinsic gas g_0 is calculated as follows:

$$g_0 = \begin{cases} G_{txdatazero} & \text{if } \{ST_e\}_d = \varnothing, \\ G_{txdatanonzero} & \text{otherwise.} \end{cases}$$

Up-front Cost: Upfront cost v_0 is calculated as:

$$v_0 \equiv \{ST_l\}_g * I_p$$

Remaining Gas: Remaining gas g for computation is:

$$g = \{ST_l\}_g - v_0$$

Slot Transaction Validity: The validity of an ST can be checked in much a similar way to regular transactions.

$$\begin{aligned} ST_l &\neq \varnothing \ \wedge \\ \sigma[\{ST_l\}_a] &\neq \varnothing \ \wedge \\ \{ST_l\}_n &> \sigma[\{ST_l\}_a]_n \ \wedge \\ g_0 &\leq \{ST_l\}_g \ \wedge \\ v_0 &\leq \sigma[\{ST_l\}_a]_b \ \wedge \\ \{ST_l\}_p &> 0 \ \wedge \\ \{ST_l\}_g &\leq B_{H1} - l(B_R)_u \end{aligned}$$

Regular Transaction Validity: The validity check of a regular transaction is changed slightly to accommodate slots. Note that in the Ethereum Yellow Paper, the address of transaction T is denoted as $S(T)$. Because S is used a lot in this document, the address of transaction T is referred to as T_a.

$$\begin{aligned} T_a &\neq \varnothing \ \wedge \\ \sigma[T_a] &\neq \varnothing \ \wedge \\ T_n &= \sigma[T_a]_n \ \wedge \\ g_0 &\leq T_g \ \wedge \\ v_0 &\leq \sigma[T_a]_b \ \wedge \\ T_g &\leq B_{H1} - l(B_R)_u \ \wedge \\ \sigma[T_a]_{lq} &= \varnothing \end{aligned}$$

Execution Environment: With the above information, an execution environment can be initialized. Once the execution environment is set up, it can be executed like a normal transaction.

- I_a, set to $\{ST_l\}_a$.
- I_o, set to $\{ST_e\}_a$.
- I_p, calculated above.
- I_d, set to $\{ST_e\}_d$.
- I_s, set to $\{ST_e\}_a$.
- I_v, set to 0.
- I_b, set to $\sigma[\{ST_l\}_a]_c$.
- I_h, the block header of the present block.
- I_c, set to $\varnothing$.
- I_w, given permission to change state.

The machine state is set up as follows:

- μ_s, set to $\varnothing$.
- μ_m, set to $\varnothing$.
- μ_i, set to 0.
- μ_g, calculated above to be g.
- μ_{pc}, set to $\{ST_l\}_c$.

8 BINDSIG and EMITSIG Opcodes

The following are the formal definitions of BINDSIG and EMITSIG. δ is the number of inputs via the stack and α is the number of items returned/added on the stack.

<table>
<tr><th>Opcode</th><th>δ</th><th>α</th><th>Description</th></tr>
<tr><td>BINDSIG</td><td>5</td><td>1</td><td>
This opcode binds a listener to a signal specified with its sigId.
It binds a new leaf to the slot trie of the emitter contract.

<table>
<tr><th>item on stack</th><th>Description</th></tr>
<tr><td>0</td><td>emitter contract address</td></tr>
<tr><td>1</td><td>sigId</td></tr>
<tr><td>2</td><td>codePtr</td></tr>
<tr><td>3</td><td>gasPriceRatio</td></tr>
<tr><td>4</td><td>gasLimit</td></tr>
</table>

if $\sigma[I_a] \neq \varnothing \wedge \sigma[\mu_s[0]] \neq \varnothing$:
 $\sigma' = \sigma$, except
  $let\ L = (I_a, \mu_s[2], \mu_s[3], \mu_s[4])$
  $\sigma'[\mu_s[0]]_l = \sigma[\mu_s[0]]_l[\mu_s[1]].\texttt{INSERT}(L)$
  $\mu'_s[0] = 0$
else:
 $\sigma' = \sigma$, $\mu'_s[0] = 1$
</td></tr>
<tr><td>EMITSIG</td><td>5</td><td>1</td><td>
This opcode creates an instance for the signal specified with its sigId. It binds a new leaf to the signal trie.

<table>
<tr><th>item on stack</th><th>Description</th></tr>
<tr><td>0</td><td>sigId</td></tr>
<tr><td>1</td><td>number of block delayed</td></tr>
<tr><td>2</td><td>pointer to signal data byte array</td></tr>
<tr><td>3</td><td>number of elements in signal data byte array</td></tr>
</table>

if $\sigma[I_a] \neq \varnothing \wedge \mu_s[1] \geq 0$:
 $\sigma' = \sigma$, except
  $\forall L \in \sigma[I_a]_l[\mu_s[0]]$:
   $let\ E = (I_a, \mu_s[0], (I_b[\mu_s[2] + 0], \ldots I_b[\mu_s[2] + \mu_s[3]]))$
   $H_{st}[I_{Hi} + \mu_s[1]].\texttt{ENQUEUE}(\texttt{RLP}(E, L))$
  $\mu'_s[0] = 0$
else:
 $\sigma' = \sigma$, $\mu'_s[0] = 1$
</td></tr>
</table>