InPay: 2-way pegged smart-contract system on the Waves platform

Version 0.2 commit `3dd424`

Exypno Kouneli

June 25, 2017

Introduction

Developers have been experimenting with decentralized applications, or DApps, since early days of decentralized ledgers. The applications were proposed both as separate blockchains, such as Namecoin [1], Datacoin [2] and several others, and as applications on top of the existing blockchains, like Omni [3] and Counterparty [4]. A big step forward for DApps adoption was made by Ethereum project [5]. General-purpose system created by Buterin, Wood, Wilcke and many others has set the way for explosive growth of other decentralized ecosystems. At the time of writing this paper, market capitalization of Ethereum platform has reached 30 billion USD, while total capitalization of DApps projects was at 3 billion USD. Main direction of DApps development on Ethereum is creating real-world agnostic blockchain contracts. In the same time many industries are experiencing the growing need for easy-to-use blockchain tokens backed by real world. With this in mind, S. Ivanov created Waves [6], a system focused on providing means to create tokens for anybody without specific technical expertise. Waves Platform presumes that blockchain applications, that are crucial for complex use case scenarios, should be developed as plugins to Waves. This paper describes the principles that allow Ethereum-compatible smart contract system to function as a plugin to Waves. This paper is organized as follows: first we describe the rules by which InPay detects transactions on Waves blockchain that should be processed by InPay contracts system or Phonon; then we observe principles of InPay transaction-based state machine. Finally we briefly describe principles of Phonon system (details are published in a separate paper [7]). We intend to develop these principles into stable features of InPay due to the fact that the project community is strong and keeps growing, despite having suffered major setbacks that could crumble other projects. Another factor influencing our choice is good distribution among cryptocurrency enthusiasts, that allows to expect steady price growth and avoid speculation and emerging price action followed only by loss of volume and interest.

InPay transaction matching

InPay plugin operation is based on the upcoming update for Waves blockchain that introduces new *DataTransaction* [8] transaction type. This type of transactions contains a data field with arbitrary length and per-byte fees. InPay plugin matches all transactions on Waves blockchain and executes those, that follow InPay transaction ruleset, in the InPay Virtual Machine (IPVM). IPVM has its own state based on its own ledger and contract objects with memory. IPVM also has its own InPay token that is equal to Waves-based InPay token and can be swapped via Phonon system.

To be detected as InPay transaction, *DataTransaction* transaction should follow this set of rules:

- Fees should be paid in InPay tokens
- First five bytes of data-field should be set to `0x494e504159` (encoded “INPAY” marker)
- Sixth byte is version byte and should be set to appropriate value. Currently supported versions are `0x01` (for InPay virtual machine), `0x02` (for Phonon) and `0xff` (miner signaling).

InPay transaction fees paid on Waves blockchain are counted during contract execution in IPVM. Thus fees can be paid partly on Waves chain via Waves-based InPay tokens and partly on IPVM chain via InPay-based InPay tokens. However, fees for *DataTransaction* transaction still should be paid fully on Waves blockchain and in contrast to Ethereum, excessive fees for InPay are not returned to sender.

Special InPay transaction with version `0xff` (miner signaling) should be processed only if sender address is exactly the same as coinbase address. Structure of miner signalling transaction:

vesrion 2-byte version of InPay plugin runned by miner node

payload arbitrary size encoded json-dictionary with signaling options.

Miner signaling transactions will be used in the future to synchronize InPay plugin updates. It should be noted that miner may not include miner signaling transaction, for instance, if not aware of InPay payload.

InPay contract system

Following Ethereum, InPay Virtual Machine is transaction-based state machine. It can be fully described by state and ruleset at any moment. Change of state is carried out by applying ruleset Υ to transaction T and previous state σ_t:

$$\boldsymbol{\sigma}_{t+1} \equiv \Upsilon(\boldsymbol{\sigma}_t, T) \tag{1}$$

Every transaction, and thus every state change, is determined and reversible in case of Waves blockchain reorganisation.

Transactions in InPay system is set of values:

nonce: integer value equal to the number of transactions sent by the sender

gasPrice: integer value equal to the number of minimal unit to be paid per unit of gas for all computation costs incurred as a result of the execution of this transaction

gas: up-front paid fees

to: the 22-byte address of the message call's recipient or, for a contract creation transaction 22-byte zero-string

value: integer value equal to the number of indivisible units of IPVM InPay tokens to be transferred to the message call's recipient or in the case of contract creation, as an endowment to the newly created account

[**optional**] **init:** an unlimited size byte array specifying the EVM-code for the account initialisation procedure

[**optional**] **data:** an unlimited size byte array specifying the input data of the message call

Meanings of fields coincide with meanings of field in EVM transactions [5].

Some differences between EVM transactions and IPVM transactions should be made:

- excesses of up-front paid gas are not returned to sender
- instead of 160-bit ethereum addresses we are using Waves addresses without hashsum (1-byte version + 1-byte blockchain id + 20 bytes hash)
- IPVM transaction doesn't contain signature (authorization is processed on transport level – Waves blockchain)

Transaction execution follows the rules of EVM. Execution environment coincides with described in [5] with the only exception: instead of Ethereum block header the environment contains information about Waves block, where `gasLimit` is set to `4000000`.

The fee schedule for IPVM is the same as EVM for simple operations:

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.

while being significantly cheaper for work with memory and other contracts:

Name	Value	Description*
G_{sset}	10000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	1000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged.
R_{sclear}	9500	Refund given when the storage value is set to zero from non-zero.
$R_{selfdestruct}$	14000	Refund given (added into refund counter) for self-destructing an account.
$G_{selfdestruct}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	20000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	10	Paid for a CALL operation.
$G_{callvalue}$	2000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	15000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	10	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	175	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	75	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

Phonon system

Detailed specification of Phonon is published in a separate paper [7]. Here we briefly describe principles of Phonon system. Phonon system uses masternode architecture to provide the bridge between Waves and IPVM blockchains. To become a masternode, an InPay node should lock collateral on 2of3 multisig (upcoming Waves feature) account with one key belonging to the node, one to a developer's account and one to an InPay community manager. Later, when more than 80% of miners operate with InPay plugin, this requirement will be substituted with locking collateral on anyone-can-spend address (address with known secret key), however miners with InPay plugin will not process transactions from this address if a masternode follows Phonon rules.

Special contract will be mined on IPVM blockchain and served as a universal multi-asset. It supports ERC20 [9] interface, adding the `'asset id'` parameter to every operation. `'asset id'` parameter coincide with asset ID on Waves. Waves assets can be exchanged to corresponding IPVM assets by sending them to one of the masternodes. Thus masternode that receive special transaction with a Waves asset should issue the same amount of asset on IPVM and send it to the address provided in the transaction. While less than 80% of miners operate with InPay plugin, masternodes should process exchanges only for the most liquid assets like

WTC and MRT, and reaching 80% threshold will allow any asset to be exchanged.

All assets received by a masternode are available for withdrawals from IPVM via special transaction, user can choose any masternode that has enough corresponding assets. Wallets should not send and masternodes should not process deposits that increase assets balance in InPay (ratio of asset to InPay tokens should be determined via DEX [10]) higher than 33% of collateral. If a masternode's portfolio value reaches 80% of collateral due to increase of some assets prices, that masternode should redistribute assets to other masternodes. Masternodes receive special fee for deposit and withdrawal operations. Additionally InPay community members are planning to create a special fund which will reward operation of masternodes. In case of malfunction (including long offline or asset portfolio spending) collateral is used to compensate for losses.

Conclusion

In this paper we propose a 2-way pegged smart contract system on the Waves platform built as a plugin to Waves node and utilizing new *DataTransaction* type. InPay implements transaction based state machine with its own transaction database, blockchain state and accounts. Since InPay state machine is based on EVM (Ethereum virtual machine), a large part of Ethereum ecosystem can be easily transferred to InPay blockchain, however some changes will be still needed to meet the requirements of Waves fees calculation. Besides that, InPay introduces Phonon: the bridge between Waves assets and special contract assets on InPay state machine. This system is based on masternodes that serve as liquidity providers, and allows to easily transfer arbitrary tokens from Waves to InPay blockchains and vice versa.

References

[1] Vinced, Namecoin repository, `https://github.com/vinced/namecoin`

[2] Datacoin website, `http://datacoin.info`

[3] R. Gross and co-authors, Omni specification, `https://github.com/OmniLayer/spec`

[4] I. Zuber and co-authors, Counterparty documentation, `https://counterparty.io/docs`

[5] G.Wood, Ethereum Yellow Paper, `https://ethereum.github.io/yellowpaper/paper.pdf`

[6] S.Ivanov, Waves whitepaper, `https://wavesplatform.com/files/whitepaper_v0.pdf`

[7] Exypno Kouneli, not yet published

[8] A. Kiselev, S. Tolmachev, S. Nazarov, git branch, `https://github.com/wavesplatform/Waves/commits/T751-data-transaction`

[9] Ethereum wiki, ERC20 pag, `https://theethereum.wiki/w/index.php/ERC20_Token_Standard`

[10] Waves platform blog, `https://blog.wavesplatform.com/what-is-waves-dex-7c311d1360a1`