

T3RN - PROTOCOL COMPOSING EXECUTION OVER MULTIPLE BLOCKCHAINS.

by Maciej Baj, v0.6.0

Abstract—We will explore the ways in which blockchain interoperability currently works and highlight perceived shortcomings; *inter alia*, no security standards in composing interoperable execution over multiple blockchains. The synchronisation of multiple chains, tangled by a single contract, remains a challenge faced by the industry. The immutability of changes which succeed within the context of single chain execution, but fail within the interoperable context, is the foremost obstacle to freely connecting multiple execution steps.

To solve this issue we propose *t3rn* (pron. [tern]), a protocol that solves the aforementioned issues by introducing a unique decentralised Circuit over the interoperable smart contracts, where execution over multiple blockchains is bound by a single transaction. The transactions executed by Circuit can compose execution on multiple chains and orchestrate results within the interoperable context. Due to multiple execution phases, changes can still be reverted until they are finally committed; binaries of smart contracts are hosted on Circuit. Developers that contribute to the *t3rn* on-chain registry have the opportunity to be remunerated for the use of the smart contracts that they develop. We demonstrate that such a network can work in a decentralised and sustainable manner, where actors follow a hybrid consensus model, which merges elements of Proof of Stake with useful Proof of Work, while security is guaranteed by Polkadot’s Parachains solution. The decentralised contracts execution platform relies mostly on the smart contracts deployed on extrinsic blockchains and only forms on-chain multi-signatures when necessary.

I. INTRODUCTION

SEVERAL blockchain systems are on the verge of introducing interoperability [1]. This creates ample opportunity for new services to emerge and support the transition from single to multi-ledger decentralised systems. We regard *t3rn* to be a second-layer service supporting interoperability for first-layer blockchains. *t3rn* also offers new features that improve the efficiency of developing interoperable systems.

When executing a smart contract on a single blockchain there are mechanisms in place that dispose of the unfinished execution effects in case an error occurs. Achieving a similar result while executing on multiple chains, that do not share a common state, is a non-trivial task which *t3rn* solves. We propose a protocol that abstracts a smart contract from a particular blockchain and elevates the validation to the external system, known as Circuit.

Smart contracts compiled with the *t3rn* toolset become composable chunks of byte code and are executed separately on appointed blockchains and can therefore be regarded as composable. The execution steps are overseen and secured by a decentralised network that follows the validation rules of the protocol. Validators collect witness and proofs delivered from

foreign chains by independent actors, referred to as relayers. Relayers observe new changes, dispatch and deliver messages between Circuit and Gateways – adapters that escrow the execution on a single chain. Execution is organised in three sequential phases: Execute, Revert and Commit, of which each may consist of many, possibly parallel, steps.

II. RELATED WORK

The design of our protocol and architecture of associated services – Circuit, Gateways and Interoperable Smart Contracts Repository is built to be highly adaptive and capable of accommodating the vast majority of modern blockchain architectures.

Circuit aligns with Polkadot’s multiple chain vision, where many independent blockchains provide a wide range of services. *t3rn* can be classified as an on-chain second-layer solution [2]. Its underlying first-layer blockchain architecture, Polkadot, communicates with Parachains – inner chains within the ecosystem – by exchanging messages using the XCMP Protocol [3]. While XCMP does guarantee the delivery of a message, it does not guarantee what code will be executed, i.e. how the receiving Parachain will interpret the message [4]. A similar messaging protocol, Inter-Blockchain Communication (IBC), has been researched by Cosmos. In IBC zones send messages and tokens to each other via the hub. As zones do not share state, a re-organization of one zone would not re-organize other zones, meaning each message is reliant on the recipient’s trust in the security of the sender [5]. Blockchains that are foreign to Polkadot’s ecosystem architectures can be reached by a Parachain using Bridges [6]. Any external system can dispatch cryptographically signed messages to any Parachain via RPC API [7]. Shared Protected Runtime Execution Enclaves (SPREE) are fragments of logic, comparable to runtime modules in Substrate, but live on the Polkadot Relay Chain, which may be opted in on by Parachains. SPREE fragments have their own storage and XCMP endpoint on each Parachain [4]. In comparison, *t3rn* keeps track of the state changes of logic execution remotely of the Parachain, on the Circuit. By doing so we ensure that all of the execution of a smart contract, as well as modification to its state, will be tracked by the Circuit to avoid implication of synchronising state changes of the same business logic across multiple chains. Execution is abstracted from data of a specific chain III-A4.

As a second-layer solution, *t3rn* finds its place above the mentioned first-layer solutions, spinning the interoperable execution using escrow account, submitting transactions via RPC

API and confirmation messages via Bridges while offering a wide range of interoperable executions, reduced to the common programming framework of composable contracts. t3rn will explore adjusting our solution to integrate with SPREE and XCMP as their development proceeds.

A. Example

The architecture of Ocean Protocol serves as a good example to analyze as it introduces a wide variety of decentralised actors implemented as Smart Contracts on Ethereum [8]. Let's lay out the context of our example: we want to algorithmically determine the amount of liquidity in ETH to provide to the decentralised liquidity provider Acala, which is running as a Parchain. That algorithm, a WASM contract registered on Edgeware's Parchain, would rely on input data fed by a data provider registered in Ocean Protocol's Marketplace, an Ethereum smart contract. We would also need an additional actor to orchestrate the whole procedure; currently there is a lack of an entity which could be used for this in a trust-free manner. As an Ethereum smart contract the Orchestrator would have to implement the following:

- integrates with both bridges on Ethereum provided by Edgeware and Acala and watches for relevant headers
- compatible with Edgeware's and Acala's encoding and cryptography standards
- acts as a special trusted actor for contracts registered on Acala and Edgeware

The example assumes that:

- Both Acala and Edgeware implement their bridges to Ethereum. If they do not, users must implement a bridge-like connection themselves.
- Orchestrator smart contract has been implemented and is deployed to Ethereum by a user
- Data access has been bought by a user and granted to the Orchestrator's account in Ocean Protocol's Marketplace [8]
- The WASM contract that receives messages from Edgeware Bridge and feeds them to the algorithm has been deployed by a user to Edgeware (fees has been paid in \$EDG)
- User has it's account on Acala, the subject to staking rewards, and integrated it in Orchestrator request to Acala Bridge dispatch.

Implementing interoperable execution could follow following steps, supported by the scheme Fig. 1:

- 1) User externally signs the recalculate liquidity transaction and dispatches to Orchestrator on ETH via RPC API
- 2) Orchestrator calls Ocean Protocol's data provider and feeds it to the Bridge with Edgeware.
- 3) Bridge receives the data on Edgeware and feeds them into the WASM contract that in turn feeds it to the algorithm, receives the output and passes it back to the Ethereum Bridge.
- 4) Orchestrator received the message from Ethereum Bridge with the newly calculated number of ETH to stake on Acala and can proceed with the final submission of that ETH amount it to Acala via Acala Bridge.

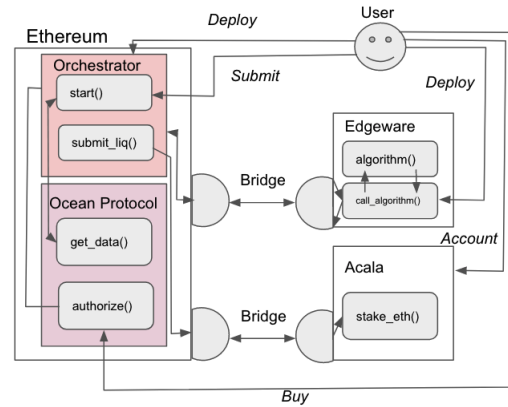


Fig. 1. Algorithmically determine liquidity using Polkadot's Parachains and Ethereum

It is important to highlight that implementing interoperable solutions is extremely complex, even assuming the Parachains in our example are production ready, which is not the case at the time of writing. In the given example of algorithmically calculating liquidity, involving three blockchains, requires a user to implement and deploy a substantially large *orchestrator* smart contract to Ethereum and a wrapper *call_algorithm* WASM contract passing the results from the Ethereum Bridge to the existing algorithm on Edgeware. In this instance it is also worth noting that a user must own three cryptocurrency wallets in order to pay for contract deployment to both Ethereum and Edgeware, as well as one for receiving staking rewards on Acala. A user will have to pay gas fees for execution costs of all additionally implemented *orchestrator* functions, calling Ocean Protocol's data contracts, execution of *algorithm* and *call_algorithm* on Edgeware and execution of *stake_eth* on Acala.

In comparison, using t3rn Circuit, a composable smart contract could be implemented as follows:

Listing 1. Algorithmically determine liquidity using Polkadot's Parachains and Ethereum writing composable contract with t3rn

```
use t3rn::{ethereum, edgeware, acala}

let data =
  ethereum::gateway_ext::ocean::get_data()
let amnt =
  edgeware::gateway_ext::algorithm(data)
acala::gateway_int::stake_eth(amnt)
```

The composable contract can be written in standard smart contract languages. After deploying to Circuit and submitting an execution request, the network ensures execution of all of the necessary steps for executing all of the smart contract components, making supporting smart contracts like *orchestrator* or *call_algorithm* obsolete. The protocol is fail-safe and insures the execution in case a misbehaviour is proven. It's also worth noting that when integrating blockchains with extrinsic programmable gateway no extra access on the part of a development team behind the project is required. Integration with t3rn can be done as a community initiative without special

consent of the chain creators, as long as their network and consensus are public.

III. PROTOCOL

We propose a protocol for composing execution over multiple blockchains, which can be described as having the following characteristics:

- **Composable** – the execution is abstracted from data and divided into multiple steps, of which any can be executed within the context of different blockchains. The protocol outlines how these independently executed steps come together as one interoperable transaction.
- **Interoperable** – provides uniformed standards that allow consensus over the results of execution on different blockchain architectures. The protocol provides many ways to integrate with a spectrum of blockchains and denotes multiple execution types possible on each.
- **Trust-free** – operated and governed by a decentralized network with a hybrid approach to consensus merging Proof of Stake with useful Proof of Work. Network security is derived from Polkadot’s Relay chain.
- **Fosters Collaboration** - binaries of smart contracts and execution stamps are maintained by the network. Submitted contracts are shared amongst the developers and the protocol benefits from a network effect, wherein its value grows with usage.

A. Composability

The t3rn protocol is capable of connecting executions on multiple blockchains. The framework allows for an interoperable transaction to be reversible, meaning the transaction is composable into numerous steps. This is one of the most important features of the t3rn protocol. As a result, an interoperable execution can be reduced to the difficulty degree of a smart contract working within a single-chain context.

1) *Execution Types*: The protocol recognises four execution types that can be ordered by requesters as a single composite on foreign chains:

- **Transfer-only Execution** involves a single asset transfer, fungible or non-fungible.
- **Pure Execution** Static, view-only.
Execution of generic programs that enforces not to cause any changes to the state of programs running on extrinsic blockchains. The pure execution prevents the state of other programs, native to that chain, to be modified and therefore is made irreversible while executing via extrinsic gateways.
Pure execution is useful when the composable contracts require read-only access to the resources on extrinsic blockchains.
- **Dirty Execution** with side-effects, irreversible on chains with extrinsic gateways.

Execution of generic programs that causes changes to the state of the programs running on extrinsic blockchains. This is the only type of execution which can be irreversible if carried through extrinsic programmable gateways. It can still be made reversible, and therefore composable, when carried through intrinsic programmable

gateways. This is due to the fact that the execution is reverted (dirty execution is identical to volatile execution in this case) with intrinsic gateways, but the effect is recorded in the form of an Execution Stamp and emitted as an event. From there, the deferred results can be applied in the commit phase via sudo access to the accounts ledger.

- **Volatile Execution** always reverted.

Execution of generic programs that always reverts its changes to the state of programs running on extrinsic blockchains. This is similar to the pure execution in the sense that it leaves no changes to the programs running on extrinsic blockchains, while it still records all of the side effects. This is particularly useful for extrinsic gateways as it gives an insight into the results of the foreign programs execution. In other words, to simulate the execution on extrinsic chains without the consequences of modifying their state.

2) *Execution Phases*: Execution types can compose one another. For every execution order, a requester creates the Composable Execution Schedule that instructs the Circuit and Agents about the order regarding which the composites will go together as one interoperable transaction.

Execution Order can group the composites within several sequential phases or parallel steps. Up until the final "Commit" phase all of the steps are still reversible.

- **Execution** – all types of execution will now be carried out by execution agents via all kinds of involved Gateways on all appointed extrinsic blockchains. Execution consists of multiple sequential phases of which all can have multiple steps. Each step must have its proof delivered by relayers and verified by the Circuit validators. The execution stamp is built up with the next successful step, and the phases can already rely and access the deferred execution results from the previous phases. This allows the execution steps to adhere their actions to the overall context of the interoperable transaction and decide whether to carry on with the execution or fail the step and start the "REVERT" phase.
- **Revert** – Optional and the last phase, triggered when the execution fails on one of the Gateways or cannot be proved by Circuit during Execution phase. During this phase each execution agent must revert the changes from the extrinsic chains involved in execution.
- **Commit** – Optional and the last phase, triggered after all execution phases are successfully concluded and all of the steps are proven to be correctly facilitated to all extrinsic chains. After validators collate a composable transaction to the relay chain, parties can claim release of their resources from the escrow accounts. Relayers are incentivized to propagate the information from relay to extrinsic chains, but any actor can trigger the release. For different gateway types the COMMIT phase means different claims:
 - Transfer-only two-way gateway IV-A3: validators co-sign the valid transactions to the execution stamp, releasing the funds on extrinsic blockchains.

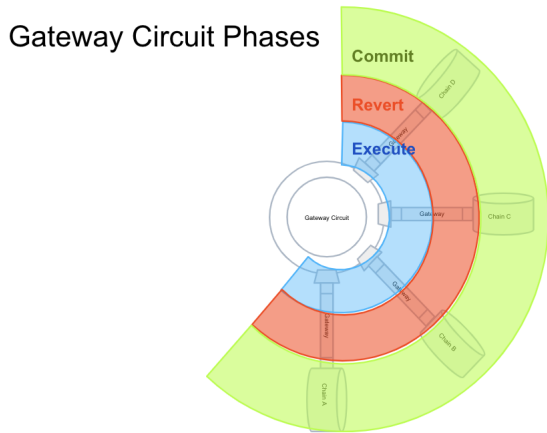


Fig. 2. Protocol follows multiple execution phases.

- Programmable gateways IV-A: funds locked on escrow accounts are available for claim and are controlled by the integrated smart contract / programs. Intrinsic gateways can now release the deferred storage write and events directly to the storage of extrinsic smart contracts programs using its special sudo authority. At extrinsic gateways, execution-dirty already left the desired effects to the foreign smart contract / programs.

After the commit phase changes to the target accounts and on-chain smart contracts / other on-chain runtime programs can no longer be reverted.

3) *Composable Compilation*: Composable contracts like the one above needs to be compiled into the executable form that would be understood by the Circuit. The output of compilation is a *package*:

- Binaries - an array of byte data containing the executable code. Array length equals the number of composable chains that the code will be executed onto.
- Metadata - a single file in JSON format containing information about the content and size of the executables. Allows for the correct distribution of those chunk into Gateways.

At this point we foresee implementation of many compilers – some of them developed as an outcome of community interest and some implemented by the development team. The compilers that are important for the sake of usability are:

- WAT (WASM) $\xrightarrow{\text{composable compilation}}$ [WASM] + Metadata
- ink! (Rust) $\xrightarrow{\text{composable compilation}}$ [WASM] + Metadata
- Solidity (Eth) $\xrightarrow{\text{composable compilation}}$ [WASM/EVM] + Metadata

Such a package is later signed by a callee and needs to be delivered to the decentralised t3rn network via either RPC or HTTP API. We support the three currently most popular blockchain architectures asynchronous signature schemes:

- ed25519 implementation using Schnorr signatures.

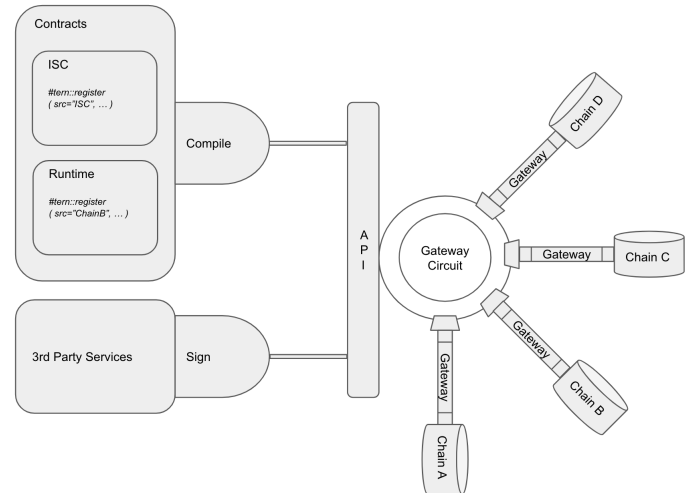


Fig. 3. Components Architecture

- Schnorrkel/Ristretto sr25519 variant using Schnorr signatures.
- ECDSA signatures on secp256k1

All of the signatures fit exactly on 64 bytes.

4) *Volatility*: Smart Contracts usually have two life-time phases:

- Deploy – instantiates the new smart contract with specified data (inside the contract)
- Call – executes that contract with specified input data

Recreation instantiates the smart contract in the given pre-state. This will be handled in the "deploy" phase of a smart contract, but instantiate the contract from the point in the history of the state where the execution finished the last time that the code was executed. The state is preserved by the circuit. In this way the circuit can stay up to date on the smart contract's state history, on each chain that it was executed.

This is an important aspect of the objective of composable for our solution. Execution is abstracted from data on specific chains. The execution within a context of an extrinsic chain can be reverted to any previous point in the history, as the underlying smart contract's state is not stored on the extrinsic chain but maintained by the t3rn circuit instead. Why recreate a contract every time on an extrinsic chain instead of leaving it there? By doing so we ensure that all of the execution of that contract and modification to its state will be tracked by the Circuit. Synchronization between the state of the same contract on the circuit and extrinsic chain is problematic and introduces unnecessary complications to the protocol.

B. Interoperable

In this section we will outline how interoperability is achieved through the open and decentralized network of the t3rn protocol. The protocol assumes that the network is always open for a new blockchain to be added: new actors to join that are incentivized to behave honestly and punished for proven misdemeanours, through slashing of their bonded stake.

a) *Foreword to interoperability*: By providing an interoperable solution we make interaction with external (extrinsic) blockchain systems possible using the same means of exchanging messages with common on-chain (intrinsic) entities. In other words, from the perspective of intrinsic blockchain entities, processing interoperable transactions looks no different than processing those signed by accounts known to that blockchain. Interoperable transactions however, trigger actions on external services by dispatching transactions on extrinsic blockchains. This raises a concern: who is authorized to dispatch the messages to extrinsic blockchains while maintaining control of funds? In the t3rn protocol, that particular entity is decentralised and open for anyone, with a bonding stake, to join. The rules of the transparent protocol are outlined in this document.

1) *Action Types*: It is important to distinguish between the execution types possible on the different blockchains involved in cross-chain execution. Currently, there are many systems that consider interoperability as a means of cross-chain token balance transfers (fungible assets) using decentralised or semi-decentralised middleman. Semi-decentralised systems rely on federated parties co-signing the transactions, locking and releasing assets on the two blockchains involved in the transfer. To our knowledge, there are very few bridges that facilitate message transfer that relies on the on-chain code deployed on both ends of the bridged blockchains. One such example being Parity Bridge, which is currently under development and cannot be considered production ready. Other known examples, like ChainSafe Bridge, include federated parties that form a consensus for attesting each balance transfer to be valid while staying in full control of the locked funds.

We do not consider this to be a satisfactory solution and have thus designed a decentralised protocol capable of a full-spectrum of interoperable actions:

- 1) Type 1. fungible (balance) and non-fungible cross-chain assets transfers
- 2) Type 2. execution of on-chain generic programs registered on extrinsic blockchains (calling smart contracts or on-chain runtime logic)
- 3) Type 3. volatile execution of generic programs (smart contracts) within a context of different blockchains: executes with a given external state and has read-only access to the programs registered on extrinsic chains. It is volatile and therefore removes itself and the assigned storage on extrinsic chains after the execution. See more in the "Volatile" subsection of the Composition section.

a) *Distinguishing between different types of blockchain integrations*: We set a framework that classifies blockchains into different types. Tailoring for that framework enables integration within the shared interoperable system. The design should be flexible enough to support all popular blockchain architectures.

The main differentiation of a type is whether it is possible to program generic logic onto a blockchain's runtime and until which point permission allows the write access of accounts ledger to non-owners with special privileges – sudo users.

- 1) **Programmable blockchains with the owner-only access to accounts ledger** These are commonly known

as smart contract (or any other generic on-chain logic) platforms. The access to the deployed smart contract is controlled by its developers and users must verify the code or trust the developer not to compromise their funds. There is no way for other entities to override the underlying smart contract storage unknowingly to developers (no sudo access).

In order to integrate t3rn with Type 1. blockchains, developers must:

- a) Deploy the execution wrapper that supports all three distinguished interoperable execution types and follows t3rn protocol.
- b) Deploy the reward and refund claims contract, that receives block headers from the Relay chain and unlocks the access to funds upon verifying the proofs. The funds deposited to the smart contract can be distributed to those with valid claims as instructed by the code, with no decentralised or federated parties involved.

Examples: Ethereum, other smart contract platforms

- 2) **Programmable blockchains + the sudo access to accounts ledger** The generic logic can be built-in into a blockchain granting full access to the accounts ledger to privileged sudo users. In order to integrate with Type 2. blockchains developers must fulfill both requirements of Type 1. Blockchains integration and additionally:

- a) Grant sudo access to federated accounts and enable writing deferred changes directly to the accounts ledger.

Examples: Polkadot Parachains with t3rn gateway pallet installed

- 3) **Non-programmable (transaction-only) blockchains + owner-only access to accounts ledger** Restricted access to generic logic, usually only balance transfer and consensus-specific transactions are allowed to be dispatched by accounts with non-negative balance on those chains. Access to those accounts is facilitated by a multisignature committee, that must abide by stringent rules in order to access funds; these rules can vary in their level of decentralisation and therefore the solution is not ideal as it does rely on the honesty of a limited amount of participants involved in the forming of multisignatures. Interoperability with blockchains Type 1. and Type 2. can be achieved by actors in the t3rn protocol having no direct access to the accounts on connected blockchains. Interoperability with Type 3. can only be achieved by actors in the t3rn protocol staying in control of the accounts of connected blockchains. This is of course a disadvantage and t3rn protocol aims to decentralize control. This can be done by "fast Multiparty Threshold ECDSA" formed by the t3rn validators that oversee the security through the introduction of bonding of stake for the multisignature participants.

Examples: Polkadot Relay Chain, Bitcoin

- 2) *Standardized*: Standards are necessary to achieve the uniformed execution on all of the smart contracts across interoperable blockchains connected within the same system - Circuit.

Standards include most of the functionalities used by a smart contract. Each connected smart contract VM or other on-chain runtime needs to implement the following standards:

- set storage (account: [u8; 32], key: [u8; 32], val: Optional [u8]) \rightarrow ()
- get storage (account: [u8; 32], key: [u8; 32]) \rightarrow Optional [u8]
- get runtime storage (key: [u8; 32]) \rightarrow Optional [u8]
- transfer (to: [u8; 32], value: u64) \rightarrow ()
- address (to: [u8; 32], value: u64) \rightarrow [u8; 32]
- caller (to: [u8; 32], value: u64) \rightarrow [u8; 32]
- balance (of: [u8; 32]) \rightarrow u64
- now (to: [u8; 32], value: u64) \rightarrow u64
- random (seed: [u8]) \rightarrow [u8]
- value transferred (to: [u8; 32], value: u64) \rightarrow u64
- minimum balance (to: [u8; 32], value: u64) \rightarrow u64
- deposit event (topics: [[u8]], data: [u8]) \rightarrow ()

IV. IMPLEMENTATION

a) *Gateways*: Execution of a contract (byte code binaries) within the context of a single chain must be secured via a dedicated wrapper – Gateway.

In theory any valid account could facilitate the execution on extrinsic blockchains ordered by requesters. In practice, only execution agents approved by the Circuit have incentive to do so as they anticipate rewards for their honest services. The circuit validators expect the inclusion and execution proofs to be delivered by the relayers. The execution on any extrinsic chain must follow the gateway execution protocol.

t3rn works on the basis of an open and decentralised system allowing participants to use different implementations of gateways, as long as they follow the protocol. Execution agents are the ones at risk of losing their bonded stake, the consequence for a gateway’s wrongful implementation. We very much encourage the community to create their own more efficient gateway implementations in the spirit of supporting a varied and effective interoperable ecosystem. We however foresee that at the beginning our team will implement the gateways to the most popular blockchains in order to allow the protocol to gain traction and market awareness.

A. Programmable Gateways

The examples of programmable gateways vary between extrinsic and intrinsic:

1) *Extrinsic Programmable Gateway*: These will be used to connect with Type 1. Blockchains - programmable with owner-only access to account ledgers. As the access to deployed smart contracts is controlled solely by its developers, the entire gateway implementation must be implemented, deployed and successfully instantiated following the deployment process specific to that particular blockchain.

To integrate a blockchain via extrinsic programmable gateway no extra access on the part of a development team behind the project is required. The t3rn development team or any other community initiative can implement the external gateway without special consent of the foreign chain creators.

Example Virtual Machine: EVM (Ethereum), Solana Runtime

2) *Intrinsic Programmable Gateway*: Execution of native byte code via the Intrinsic Gateway becomes an integral part of that blockchain and is maintained and executed by its operators. This can only be achieved through the inclusion of the gateways by development teams, while allowing special sudo access to the accounts ledger. The intrinsic programmable gateway offers the full spectrum of t3rn features for interoperable execution by making all of the execution types possible and reversible (including storage writes, thanks to the sudo access to accounts ledger).

Example Virtual Machine: Versatile Wasm VM for Polkadot Parachains - both Parachains with Contracts Pallet and without are supported.

a) *Tracking State*: Implementation that tracks the latest t3rn headers, which are submitted to the relay chain bridge and the on-chain logic that verifies that headers have been finalized. This assures integrity within the Circuit, updates the recent validator set, facilitates rewards and refunds claims in a trust-free manner, with no single entity being in control of the funds.

b) *Execution Escrow*: Execution of smart contracts (or other binaries) within a context of a single chain must be secured via a dedicated wrapper – escrow accounts. That is necessary to guarantee that the escrow accounts belong to active validators, that risk losing their stake in case of misbehaviour. Gateway executes the binaries in behalf of the escrow accounts owned by validators in order to hold off the execution results reaching target destinations up until the last COMMIT phase.

Listing 2. Programmable gateway execution flow via escrow account

```

CASE "execution":
1. Recreate attached byte code with given state (requirement of composability)
2. Emit EVENT_RECREATED with the instantiated smart contract identifier
3. Execute the recreated smart contract (or runtime module) with given input.
   // Allows the forward calls to contracts registered on that chain.
4. Record all of the fungible or non-fungible transfer throughout
   the execution to deferred transfers.
5. Revert transaction after successful execution.
   // Specific event structure submitted as a part of the protocol.
6. Emit EXECUTION_SUCCESS event with encoded EXECUTION_STAMP as a parameter.
CASE "execution_dirty":
5. Do not revert transaction after successful execution
CASE "execution_pure":
3. Enforce the state of extrinsic contracts (recreated smart contract
   from attached code can still be modified) to stay unmodified.
   // Like ("staticcall" for EVM)
IF intrinsic programmable gateway:
4. Record all of the deposited events, storage writes
   throughout the execution.
CASE "revert"
1. Invalidate deferred results.
CASE "commit":
1. Release deferred events recorder during execution
2. Release deferred transfers from escrow account to target
IF intrinsic programmable gateway:
3. Apply deferred storage writes

```

c) *Execution Results*: Since t3rn facilities not only interoperable asset transfers (type 1. interoperable feature), but also generic programs interoperable execution, each on-chain execution can result in following:

- 1) output result
- 2) storage writes
- 3) balance transfers
- 4) dispatched events (or logs)

Programmable Gateway captures all of the above results in a data structure we call *Execution Stamp*. Each execution ends with generating the execution stamp which must be relayed by a validator to the decentralised Circuit. There validators check its integrity and validity. Both valid and invalid Execution

Stamps are stored by collators, so that it is possible to claim a refund for invalid execution.

Execution Stamp are distinct by the 32 bytes of Blake2 hash: $blake2(encode(callee) + encode(time) + input + codebytes)$.

To validate the stamp circuit collects the witness – proof of inclusion delivered by relayers. Execution Stamps generated by all type of gateways (both Extrinsic and Intrinsic) is however uniformed and produced by the following rules:

- (a) output result = $blake2(execution_output)$
- (b) balance transfers =
 $Array < encode(from : str, to : [u8, 32], val : u64) >$
- (c) dispatched events:
 $Array < (topic : str, addr : [u8, 32], input : [u8]) >$
- (d) storage writes (for *Extrinsic Programmable Gateway*):
merkle_tree.root of
 for_each write in storage_writes:
 merkle_tree.insert
 blake2(enc_key, enc_value)

3) **Transaction-only two-way Gateway**: This non-programmable type of gateway makes it possible to facilitate transfer-only (fungible + non-fungible) transactions through it. The execution, in respect to the protocol and its execution phases can still be reverted, so that a requester could get their assets back in case an interoperable transaction fails.

Transaction-only gateways do not require implementation on the side of foreign chains as an opposite to the programmable gateway. However, this does come with the cost of the multi-party signature formation for t3rn validators. From the perspective of a foreign chain, the gateway looks just like any other regular account.

Transaction-only Gateway is two-way, that means that it comes with two components connected. Circuit validators generate valid transaction (signed by multisignature) which is included in a Circuit block and can be relayed by any actor to a foreign chain. Relayers dispatch signed transactions from foreign chains to the circuit (execution orders) and the other way around (claims and refunds). On the circuit, an execution agent is chosen to facilitate the transaction. After fulfilling its duty validators generate a transaction with a reward transfer for the execution agent and co-sign it. These are valid on foreign chain transactions that transfer an amount deposited on the gateway by a requester. Similar transactions of a refund to the requester are created and co-signed by validators in case of dishonest behaviour on the part of the execution agent.

Listing 3. Transaction-only on-gateway execution flow

```
CASE "execution":
  // Transfer fungible or non-fungible assets
  // from requester to escrow account
  // and add to deferred transfers
CASE "revert"
  // Cancel deferred transfers from
  // escrow account back to the requester
CASE "commit"
  // Release deferred transfers from
  // escrow account to target destination
```

That gateway is generally designed to connect t3rn Circuit with Type 3. transaction-only blockchains, like Polkadot Relay Chain. That does not mean however, that blockchains with

extended capabilities to add the on-chain programmability, like Ethereum, cannot integrate the transaction-only gateway as well. This is further explained in the following Circuit paragraph.

B. Circuit

Circuit is an implementation of t3rn protocol maintained by validators and collators. It keeps the global state needed for interoperable execution. Being named after the electrical circuit is not a coincidence; the design is inspired and modelled by the concept, facilitating the connection of many different elements and transports signals (messages) between them. With that perspective, Circuit is a microprocessor analyzing received signals from Gateways, an adapters for foreign blockchains.

1) *Validation of Execution*: To verify whether a step of interoperable execution has been successful, the code is re-executed on the Circuit and results are compared with the execution stamp received from Gateway data. Circuit validates the execution by collecting the appropriate witness of each foreign chain (delivered by relayers):

- block header pre-execution
- block header post-execution
- block post-execution events

Circuit validators re-execute each execution step. Having smart contract binaries and input the results of re-execution are compared with collected execution stamp and its integrity checked with witness.

2) *Registry*: Registry of all of the protocol rules, actors, services and contracts involved in composable execution. Registry is maintained by collators.

- Composable Contracts – each successful execution adds a new contract to the composable contracts. These can be re-used by other developers to help them in building their own decentralised services. Contracts can also be submitted into the registry with no prior-execution.
- Chains – chains need to be pre-registered in the registry in order to provide the information about their Gateways, which in turn provide the information about fees (to calculate execution costs) and standards available to be called on that chain alongside the output parameters that are necessary to form execution and inclusion proofs.
- Statistics – statistics about actors of the system that can serve as a base for the reward campaigns instigated by the community via governance.

V. TRUST-FREE

In this subsection we will outline how to compose the execution of transactions over multiple blockchains in a trust-free manner; we will describe an open network for non-federated actors to join and be incentivized to provide honest and high-quality services.

The decentralised t3rn network is designed to fit into Polkadot's Parachain framework and will become part of Polkadot by leasing a parachain slot. This underlies the following proposal on how the network's economy is arranged with stipulation to the open-market service providers with zero-block rewards. Such a construct is possible from the point of

blockchain security, as the decision on branch validity is made by Polkadot's Relay Chain. The Relay Chain also provides the Parachains with a finality gadget. Parachains can lease the security of the Polkadot network by bonding DOT for the parachain slot.

A. Actors & Roles

All of the source codes and binaries will be actively maintained and published by the team so that there are no restrictions on how many network participants there must be. To provide a secure framework over the composable execution we distinguish additional actors within the system and design economy to incentivize open participation within it:

- **Requesters** create, sign and submit the interoperable transaction for execution. Requesters can either submit the order directly to the Circuit or dispatch it via Gateway. Expecting the interoperable transactions to be executed and influencing their priority, the requester also pays the execution fee that should cover the execution costs i.e. collation, validation and relay of necessary data for the transaction. Fees can be dynamically adjusted depending on network overload and demanded priority.

- **Interoperable Transactions** contain all of the necessary data for interoperable execution via Circuit and must be signed by a t3rn account with sufficient balance to cover fees to be considered valid. If a transaction is dispatched by a requester via Gateway on a foreign chain, it must have the following attributes assigned to the message; this means that all blockchains that implement extrinsic programmable or transaction-only gateway must support a generic opt code attached to a transaction, e.g. "data" field for Ethereum tx. That optional message contains:

- code hash – contract address on Circuit
- input – optional input for that contract
- input length – length of input data bytes

The execution of interoperable transaction has the following lifecycle:

- 1) Either requester submits a transaction for execution to the native chain A and relayers submit it to the Circuit.
 - 2) Or requester submits a transaction for execution directly to the Circuit.
 - 3) Execution Agent is being selected as per the rule described in "Selection Method".
 - 4) Execution Agent facilitates the execution on appointed chains, following the execution schedule outlined by the requester. It risks its own stake and acts as an intermediate escrow account securing the execution.
 - 5) After execution of each step is completed and the transaction is confirmed, with proofs, by the Relay Chain, Execution Agents can claim the funds deposited by requesters on native chains via gateways.
- **Collators** are the maintainers of t3rn network resources Circuit operators. They act as a collator as per Polkadot's Parachain nomenclature, maintaining a full-node of the parachain, retaining all necessary information of the parachain and producing new block candidates to pass to the Relay Chain validators for verification and inclusion

in the shared state of Polkadot. Collators store pending interoperable transactions in a pool and match the proofs provided by relayers with attestations of execution agents. Collators produce block candidates and submit them to the Relay Chain. Collators include the block reward for their services paid in \$t3rn token, which is set by the Requester.

- **Validators (Execution Agents)** facilitate interoperable transactions between the Gateways. Execution Agents can expect rewards (depending on the operating Selection Method) and collect execution fees paid in \$t3rn, covered by Requesters.

Validators compete with one another to provide attractive services and fee rates and risk losing a part of their bonded stake for not following the protocol; it is the responsibility of a fisherman or collator to report misconduct. Accounts on extrinsic chains used by agents to facilitate the interoperable execution are called escrow accounts, from the perspective of the protocol. Collators maintain the registry which holds the list of all known escrow accounts. Any observed misbehaviour, proven by fishermen, that compromises requesters' funds is subject to penalties. Stake bonded by agents must always be x2 higher than the value of the transaction that they facilitate. Execution Agents registering their services must control accounts on at least two chains and submit their addresses to the registry. Agents with an account active on every chain can compete for every single phase execution of every transaction. That predicates that each agent needs to maintain their own account which will serve as an escrow on every extrinsic chain that the transaction involves. As Execution Agents attest with their stake for the validity of transactions, their incentive is to validate a transaction before submitting it to the collators.

- **Relayers** observe changes to the target and escrow accounts on foreign blockchains involved in the interoperable execution and relay the proofs of correct execution and inclusion witness in the Circuit.
- **Fishermen** report on observed misbehaviours to Collators. Collators have authority to include the misbehaviour proofs within the transaction, slashing the bond of validators accordingly. **We set the rule that the bond staked by each validator needs to be at least 2x of the total worth of any transaction the validator executes.** Requesters are therefore insured against the balance being wrongfully spent by escrow accounts controlled by validators. Fishermen get 10% out of the violated sum that they are reporting.
- **Nominators** stake their \$t3rn for validators. In exchange validators share their gains with nominators. This incentivizes community members to be active within the t3rn ecosystem and adds utility to the \$t3rn token model. This actor is introduced only in Development Phase II. Before this phase, the Execution Agents compete based on open market rules.

B. Leader Selection Method

In this subsection we consider two methods for solving the problem of having multiple service providers for a single task. Each actor within the system is assumed to be plural. For achieving a consensus on selecting just one deputy we see two methods that would work for the t3rn protocol.

1) *Open Market*: The first method is an competitive open market; no extra rules are needed to select a leader. Double-spending prevention mechanisms are standard for decentralised projects therefore all extrinsic blockchains are assumed to have a way of selecting only one transaction out of many valid ones that then becomes recorded on the blockchain.

2) *Golden Ticket*: The participant selected as task leader receives a so-called "golden ticket". An open race is therefore reduced to a single deputy performing a task on behalf of the whole collective. This selection method is more complex and parties must follow a common algorithm of selecting a leader facilitating a single task. The specifics of the algorithm depend on the operative economy system. Such an algorithm can unintentionally be prone to a variety of Denial of Service attacks on the individual nodes of a blockchain network, as its IP must become public knowledge for the sake of the correct functioning of decentralised peer-to-peer message exchange. That problem has been analyzed thoroughly while designing modern Proof of Stake systems like [9]. The problem should not be overlooked since there is fierce competition between coins, as well as potential gains from short selling. Initiatives like Blockchain DoS (BDoS) take advantage of it and introduce "incentive-based DoS and unlike classical DoS, BDoS targets the system's mechanism design: [...] it exploits the reward mechanism to discourage miner participation" [10]. We therefore use Blind Assignment for Blockchain Extension (BABE) [11], as an algorithm to elect the leader that receives a golden ticket (either collate a block or act as an escrow during particular interoperable execution). BABE's election is based on verifiable random function (VRF) of validators invented et al. for Ouroboros Praos [9]. If a VRF output of a validator is less than a pre-defined threshold, then the validator is legitimate to receive a golden ticket [12].

Actors will have the following method to select a single deputy for a task:

- Requesters – Open Market
- Collators – Open Market transition to PoW-like Golden Ticket
- Execution Agents – Open Market transition to PoS-like leader selection
- Relayers – Open Market
- Fisherman – Open Market

The transition is further explained in the Development Path subsection V-C.

C. Development path

We foresee a transition of the implementation of selecting a leader amongst the collators and validators from an open-market to decentralised Proof of Stake. The open market rule, being an easier one to implement is a great choice for initial launch of the product. Initial launch can therefore assume

the Open Market rule for validation (execution) and collation services with zero additional to fees rewards.

We envision t3rn changing its underlying decentralised network model after initial launch. We plan to launch the network on a semi-decentralised Proof of Authority model and transition the network to a fully decentralised model after the community is engaged and the product is fine-tuned.

The date of transition to the next step also relies on the date and availability of a Polkadot Parachain slot. Migrating from a Proof of Authority model too early, before receiving guarantee of a Polkadot Parachain slot could compromise the Circuit's security.

Our plan is to develop the network in three phases:

- 1) Phase I. Release the initial network running on a semi-decentralised Proof of Authority model.
- 2) Phase II. Introduce an execution agent selection mechanism amongst the registered validators working on a Proof of Stake premise.
- 3) Phase III. Introduce a golden ticket selection mechanism for a single collator amongst the open market of competing collators, working on the Proof of Work premise. Proof of Work is made useful by computing the validation proofs instead of a math puzzle.

The transition between phases is based on community involvement and their opinions in regards to the next protocol phases will be determined by on-chain governance.

Note, that transition between Phase II and III might not even break the protocol. Validators can still submit their attestations of valid transactions directly to the relay chain. They will however most likely be rejected as the chances of obtaining the golden ticket at random are relatively low.

1) **Phase I. Proof of Authority**: is transition-only, temporary, phase. The network is secured by federated authorities responsible for keeping security intact. The incentives to become a federated validator are out of intrinsic interest in maintaining the project. There are no rewards for running the federated validation services. Initially, we will run a significant portion of validators ourselves with plans to later distribute the authority to prominent partnership projects and active community members. The parties with federated authorities will have no privileges in the next fully decentralised phase.

During that phase, the t3rn Parachain is implemented as a public permissioned blockchain over the smart contract execution platform. The economy for execution agents is designed as a competitive open market for execution services, with open and equal rules for providers to participate, regardless of their % in the stake. Zero block reward approach eliminates shortcomings observed in the Proof of Stake systems, where the "rich get richer" and actors with a higher stake are more likely to form "cartels", which can centralize the actual control over data being submitted to the blockchain.

There needs to be at least one honest actor in each category for the system to function correctly. That is enough to collect at least one correct proof per each step of Composable Transaction and deliver it to the Relay Chain validators.

D. Hybrid Approach to Consensus

Our vision of the protocol is to provide a thriving economy that incentivizes actors to compete in maximizing validation security, with large community involvement and voting for competitive validation and execution agents present in the Proof of Stake system. Proof of Stake has many merits in regard to community engagement and adding the utility of governance to security. We will therefore design a fully decentralised, hybrid consensus protocol that takes what is best from the Proof of Stake and Proof of Work systems.

1) *Phase II. Proof of Stake for Execution Agents:*

a) *Foreword to Proof of Stake:* In Proof of Stake systems, decentralised validators form a collective consensus over data, selecting a leader who has permission to add new data to the system. They act as a security guarantor for external actors and for claim rewards for their services. New active validators can be selected by being voted in. The more stake is voted in favour of a validator, the higher chance it has to be selected as a leader. Validators usually risk their bonded stake, which can be slashed for a proven misbehaviour. Individual stake holders are incentivized to nominate and vote their with their stake for individual or groups of validators. The incentivization is realised either on-chain, off-chain or both, when validation rewards are shared with nominators.

b) *Execution Agent Selection:* Validators form a collective and arrive at consensus in selecting an individual deputy that executes a transaction on behalf of the whole group.

Out of the collective of validators, it is beneficial to select a single representative that facilitates a given transaction. This is due to:

- avoiding a race for dispatching a single transaction by many parties on extrinsic blockchains.
- performing multiple transactions simultaneously by validators by deputing multiple agents.
- a collective of agents have a bigger collective stake and therefore being able to offer better services for requesters, namely higher liquidity and lower fees.

The algorithm weighs the agent's stake and historical performance. The algorithm has a collective random number generation method and the new number is drawn for each task selection. The more stake an actor has and the better a performance history, the more chance it has of getting selected for each draw.

The selection of execution agents amongst the validators can be approximated with the following steps:

- 1) Collators receive an interoperable transaction request.
- 2) One or many validators volunteer for execution of the transaction. All of the volunteering validators must have native accounts active on extrinsic blockchains involved in the interoperable execution (escrow accounts).
- 3) Validators use the selection algorithm to collectively choose a leader for the transaction - an execution agent.
- 4) Execution agent executes the transaction.
- 5) Validators attest for the transaction's correctness. If a transaction is proven not to be executed correctly, the requester can claim indemnity from stake bonded by the validators, directly from the collators.

- 6) Validators issue a reward for an execution agent.

Additional Roles of Validators Validators must achieve consensus about the selection of an execution agent for a single execution of a transaction and submit the completed transaction to Collators (Development Phase III) or directly to the Relay Chain (Development Phase II). Validators are now Collators as well.

2) *Phase III. Useful Proof of Work for Collators:* Useful Proof of Work can be a subtle extension on top of the t3rn protocol, incentivizing collators to prove their correct validation of interoperable transactions by providing the computation proof. The proof yields a unique identifier. If the identifier satisfies the current requirements of the lottery, the collator producing it receives a so-called golden ticket. A collator with the golden ticket is authorised to dispatch the transaction and its supporting computation proof to the Relay Chain and get a reward. This adds significantly to the security of the overall design, where each transaction step is checked a multitude of times by multiple competing collators.

a) *Concerns surrounding Proof of Stake:* The main concern we see in arranging a network on pure Proof of Stake premises is the lack of assurance that the transaction was actually re-executed by validators. Validators attest with their stake for the correctness of the execution and requesters can claim the x2 refund if the misbehaviour is proven by fishermen. Validators could however simply guess the correctness of the transaction without actually checking it if fishermen fail their duties.

Forming pools of validators can be regarded as inevitable in decentralised systems. In Proof of Stake systems pools with the majority of voting power exert disproportionate control over the blockchain. In practice, that system works well for popular projects with a majority honest community, that holds dishonest actors accountable. For the smaller projects however, individuals and groups with a large percentage of capital invested hold an immense amount of power and in practice have sole control of the blockchain. As mentioned, pools could have great effect for the end users if incentivized correctly. Validation pools can offer greater liquidity and high quality services for interoperable transaction requesters, as collectively they manage larger sums. In t3rn we assume the formation of validation pools as a natural evolution of decentralised projects. We add additional security layers on top of the validators to mitigate the security concerns and make no distinction between pools and individual agents, as long as they follow the protocol and convince requesters to facilitate execution via the services that they offer. A particular pool can be corrupt, but as long as the protocol is flexible enough to provide an alternative pool, there is no incentive for the pool to provide dishonest services. In the hybrid approach we propose, we put no boundaries over how validators form themselves within the groups, as long as they follow the common algorithm of selecting a single deputy per execution collectively (explained in Proof of Stake / Execution Agent Selection).

b) *The useful Proof of Work Algorithm:* To prove that the transaction was validated correctly we therefore introduce the useful Proof of Work over the computation proofs. The

algorithm selects a single collator amongst the competing group - a “golden ticket” winner. The probability of a collator becoming a winner does not rely on the stake it has, but the amount of running resources they use for validation instead. The algorithm should take some time to generate the golden ticket. That time difference will allow for the winner to have permission to perform a task individually and for the account to be rewarded.

The algorithm for the golden ticket lottery starts by receiving an interoperable transaction attested by validators to be correct. Collators look for the winning validation proofs. The proof consists of:

- unique hash identifier
- computation log
- interoperable transaction
- random nonce for which the unique hash id yields with a golden ticket
- execution stamp

The algorithm runs as follows:

- 1) Pick a random nonce from 0 to u64.MAX
- 2) Run a t3rn-compatible VM for each validation step. The VM takes the nonce as input and produces the trace level debug logs during validation. The VM prepends the next log messages with the nonce. Correctly instrumented VMs will be provided by the team, but there is no requirements on the specific instance that must be used, as long as the produced logs are consistent.
- 3) Calculate blake2 hash of the computation log. In order to calculate the unique hash id of the validation proof concat the hash of computation log, encoded interoperable transaction, execution stamp and calculate the blake2 hash of it.
- 4) Check whether the calculated unique validation hash satisfies the requirements of the golden ticket lottery. The output hash needs to be smaller than a certain number encoded on 32 bytes. This number will be dynamically adjusted based on the current network difficulty, in a similar way that the algorithm implemented in Bitcoin.

VI. ECONOMY

Each actor in the system should be incentivized to provide honest contribution and maintain the healthy network.

Types of incentive include either execution fees and rewards:

A. Execution Fees

Fees are essential to incentive for the network to provide their execution services, as there are no block rewards.

The pure cost with no rewards for execution services sets the minimum of what validators need to spend to function without losses:

$$\sum_{n=1}^{N_{chains}} gas_price(n) * gas_spent(n) + exchange_fees$$

To incentivize the services foregoing the execution, requesters assigns the fees as they see fit, based on the ongoing network load and the priority that they demand. The more steps that are involved in a Composable Transaction, the more fees will

generally be necessary, as the fees split equally between the phases for all actors involved. Additionally, depending on the total value that is requested to be transferred between chains validators might put an extra fee on top.

The ecosystem receives its share as the collators are stipulated by the protocol to account an additional 10% as an ecosystem fee used to develop and maintain t3rn. The funds will be managed in a similar model proposed by Acala introducing a decentralised Sovereign Wealth Fund [13].

$$fee = \sum_{n=1}^{N_{steps}} 1/N * (collator_fee + validator_fee + relay_fee) + ecosystem_fees + total_cost$$

Let’s take an example a request for a transfer of value 10 \$X from one chain to target account on chain Y with $total_cost = 0.2\$X$ (this is independent of t3rn protocol so far and only covers cost of operations on extrinsic chains). Requester sets the additional rewards for facilitating that execution to:

$$collator_fee = 0.03 * total_cost, validator_fee = 0.06 * total_cost, relay_fee = 0.01 * total_cost$$

In total the $service_fee = 0.1 * total_cost$ and $ecosystem_fee = 0.1 * service_fee$.

Cost of executing this transaction is set to be: $total_fee = 1.11 * total_cost = 0.222\X .

B. Execution Agents Provision

The protocol incentivizes providing validation and execution services by sharing the execution fees. Execution agents are responsible for managing their stakes in native cryptocurrencies. Staked bond is always x2 more then worth of composable transaction they volunteer to execute.

The life cycle of registering new validators is as follows:

- 1) Register validation services to the registry submitting bonding stake and details on minimum rates below of which transactions won’t be assignable for execution.
- 2) Register validation services to the registry submitting bonding stake and details on minimum rates below of which transactions will not be assignable for execution.
- 3) Provide Liquidity to the pool offering exchange of multiple pairs $\$t3rn - \$X - token$ from the same account that submitted to the registry. Get paid in $\$t3rn$ proportionally to the liquidity provided.

C. Staking

Staking supports the operations of t3rn as a decentralised hosting platform for smart contracts by supporting its liquidity (volume and size of interoperable transactions), accessibility (encourages adding new tokens on native blockchains) and in the later development phases multiplies the security factor by introducing a useful Proof of Work. Collators compete for rewards by generating validation logs of interoperable transactions.

1) *Governance Voting*: Community plays a role in staking, governance and voting. This helps to maintain a healthy ecosystem and adapt to quickly to changes within the industry. The community should be incentivized to maintain a healthy and efficient ecosystem, as each account voting on execution

agents gets a share of their rewards. Any \$t3rn token holder can submit their vote transaction as part of Circuit on-chain governance system.

2) *Staking on Validators*: Token holders can delegate their \$t3rn accounts to a validator or validators pool, who will be in a stronger position to be selected as transactions executioners. The user can then earn part of the rewards generated by validators.

3) *Liquidity Provision*: Token holders can also directly stake their native cryptocurrencies to liquidity pools operated by t3rn validators on native blockchains – Gateways. Rewards for providing liquidity in a foreign cryptocurrency get paid in \$t3rn token proportional to how much liquidity they have provided, and are distributed each time a transaction is facilitated using that pool.

REFERENCES

- [1] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, “A survey on blockchain interoperability: Past, present, and future trends,” *Instituto Superior Técnico, Universidade de Lisboa, Portugal*, 2020.
- [2] Binance, “Layer 2,” *Binance Wiki*, 2019.
- [3] W3F, “Xcmp overview,” *Research at W3F*, 2020.
- [4] Polkadot, “Spree,” *Polkadot Wiki*, 2020.
- [5] —, “Polkadot and cosmos,” *Polkadot Wiki*, 2020.
- [6] —, “Learn bridges,” *Polkadot Wiki*, 2020.
- [7] —, “Node interaction,” *Polkadot Wiki*, 2020.
- [8] O. Protocol, “Hitchhiker’s guide to ocean protocol,” *Blog Ocean Protocol*, 2020.
- [9] B. David, P. Gazi, A. Kiayias, and A. Russell, “Ouroboros praos: An adaptively-secure, semi-synchronous,” 2017.
- [10] M. Mirkin, Y. Ji, J. Pang, A. Klages-Mundt, I. Eyal, and A. Juels, “Bdos: Blockchain denial-of-service,” *CM SIGSAC Conf. Comput. Commun. Secur.*, 2020.
- [11] H. K. Alper, “Babe,” *Research at W3F*, 2020.
- [12] W3F, “Block production,” *Research at W3F*, 2020.
- [13] Acala, “Decentralised sovereign wealth fund,” *Blog Post*, 2020.