CONTRACTLAND: A FRAMEWORK FOR HIGH PERFORMING APPLICATION-SPECIFIC BLOCKCHAINS

Draft 0

ContractLand Foundation

team@contractland.io

www.contractland.io

Abstract

Present-day blockchain technology have shown promise as infrastructure for pseudonymous online payments, cheap remittance, trustless digital asset exchange, and smart contracts. However, public blockchain systems suffers from a number of issues in scalability and extensibility limiting their potential in supporting applications with critical mass.

In this paper we present a framework for solving blockchain scalability using high performing application-specific blockchains. Scalability is addressed on both the consensus layer as well as the interchain communication layer, enabling a parallel scalable architecture.

1. Introduction

Blockchain technology popularized by Bitcoin was revolutionary in enabling anyone to own and transfer assets across an open financial network without the need for a trusted third party. Ethereum further popularized blockchain with smart contracts. The concept extends the capabilities of blockchain technology to a much higher degree, disrupting the fundamental form of human trust and interactions where anyone can use an open network to run program and contractual relationships through code. However, despite the

technological promise and grand talk, we have yet to see a significant real-world deployment of present blockchain technology. We believe that this comes down to three key failures of present technology stacks:

- **Scalability**: Excessive resources are spent on the processing, bandwidth, and storage of blockchain systems. Transactions cannot be efficiently processed under peak conditions.
- **Isolatability**: A single blockchain system cannot simultaneously meet the needs of multiple applications to an optimal degree.
- **Interoperability**: Values within individual systems are closed. Systems are unable to communicate with one another.

Current real-world blockchain networks are practically limited to around 30 transactions per second. This is most apparent in Proof of Work (PoW) systems such as Bitcoin and Ethereum. The main problem lays in the leader election process, which is performed randomly and results in a significant period of system freeze. More efficient blockchain implementations such as the EOS [1] running on Delegated Proof of Stake (DPoS), or Ethereum [2] running on Proof of Authority (PoA) can process in excess of 3,000 transactions per second on performant consumer hardware. The improvement is made by decoupling the blockchain's operation of leader election and transaction serialization into separate planes. Leader election is done in a timely fashion where time is divided into epochs, with each epoch having a single leader.

However, a system described above is still insufficient to support any significant volume. A modern web based application can easily reach thousands of transactions per second. But since the processing power on public networks is shared amongst all applications on the network, a single popular application could congest the entire network for the rest. The notion of a single chain to rule them all is unrealistic. The way modern day public blockchains are built as a "world computer" are not suitable for running large-scale

applications. It is not enough to improve on the performance of blockchain systems by means of optimizing the consensus mechanism. A fundamental shift on how blockchain systems should be deployed to serve real-world applications is needed.

It seems clear, therefore, that one reasonable direction to explore as a route to a scalable decentralized computing platform is to out-scaling parallely. We decouple the application layer from the consensus layer, allowing individual application-specific blockchains (or app-chains) to be deployed in isolation while enabling the communication between them.

2. Summary

ContractLand provides a general technology framework designed for building blockchain systems to support critical mass. This mean we could use this framework to deploy application oriented blockchains in alternative to building applications on public blockchain networks. An app-chain should have comparable security and decentralization properties of a public blockchain while being significantly more efficient as the network's processing power is dedicated to a single application domain.

2.1 System Overview

In order to build a high performing public ledger capable of supporting high volumes of transactions, we need to tackle the problem of decentralization, scalability, and interoperability, to do so, ContractLand is building from the ground up. Starting with the blockchain layer, to the cross-chain communication layer, and all the way to the application specific logic layer. Each layer is designed to be modular and minimal in logic.

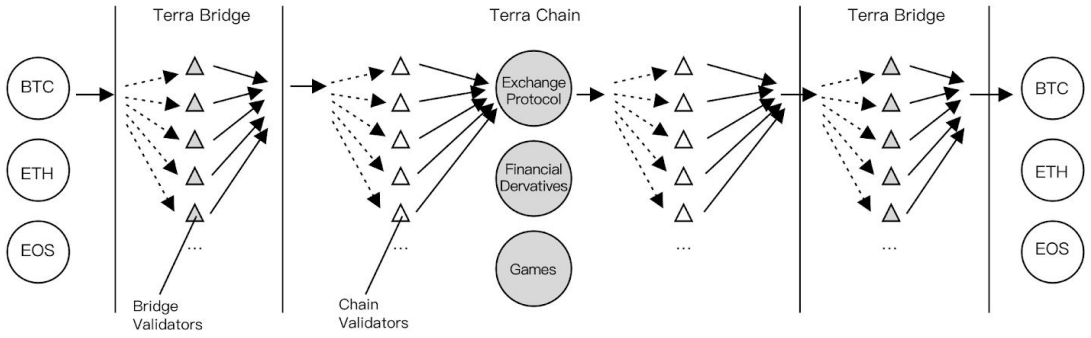

Figure 1. Overview of the application-specific blockchain framework

2.1.1 Chain Layer

Differing from existing public blockchains, ContractLand's blockchain layer is designed with scalability in mind to support critical mass. It does not implement any inherent functionalities, and offloads the application specific logics to the application layer. Unlike existing blockchains such as Ethereum or EOS, ContractLand's blockchain does not allow arbitrary logic execution, but will be dedicated to a single application domain. This is a conscious design decision to reduce transaction throughput dilution that exists in public blockchain systems where transaction throughput is shared by many different applications. However, given each layer is modularizable in design, different application logics could parallelly scale out by deploying on a large number of app-chains that are interconnected by cross-chain bridges.

2.1.2 Bridge Layer

The bridge provides the ability to perform interchain communication in a multi-chain ecosystem. This is a critical component of the framework as it not only allows individual app-chains to communicate with each other, it also opens up the ability for them to communicate with other existing blockchain networks. Just like the blockchain layer, the bridge has its own consensus and incentive mechanism, ensuring the secured and decentralized inter-chain transactions.

2.1.3 Application Layer

The application layer contains application specific logics written any EVM compatible language such as Solidity [3]. Due to the modularity of the system, the application logic is not limited to any particular set of constraints, and could be applied to a large array of domains such as exchange, payments, finance, and games.

3. Participants of ContractLand

There are 3 basic roles in the upkeep of an app-chain in the ContractLand framework:, chain validators, bridge validators, and delegators.

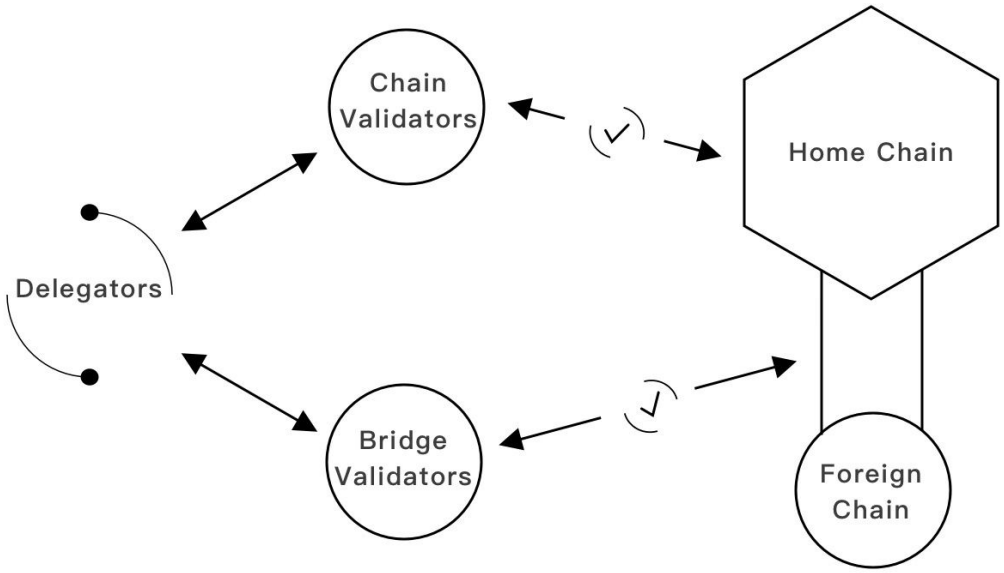

Figure 2. Relationship between the roles within an app-chain

3.1 Chain Validators

A chain validator is the highest charge and helps seal new blocks on an app-chain. The validator's role is contingent upon a sufficiently high bond being deposited, though we allow other bonded parties to delegate the role to other validators to act for them and as

such some portion of the validator bond may not necessarily be owned by the validator itself but rather by these delegators.

A validator must run a chain client implementation with high availability and bandwidth and maintain the proper functioning of the blockchain network. This process involves receiving, validating and publishing candidate blocks. Validators are also responsible for approving foreign chain bridge integrations. While the bridge to connect with a new blockchain system can be develop and proposed by the core development team, is it up to the validators to verify and integrate the bridge as part the app-chain.

A validator not fulfilling their duty to find consensus under the rules of our chosen consensus algorithm is punished. Benign behaviours of validators will result in the reduction of their security bond. Provably malicious actions such as double-signing or conspiring to provide an invalid block result in the loss of the entire bond. For both benign and malicious behaviours, the penalized stake is partially burnt and mostly given to the informant, which are other honest validators.

In some sense, validators are similar to miners of current PoW blockchains.

3.2 Bridge Validators

Bridge validators play a similar role as chain validators but for bridges. Each bridge in an app-chain has its own consensus and security ecosystem maintained by it's bridge validator set. Just like chain validators, a bridge validator must run the bridge client and necessary corresponding blockchain nodes for relaying cross-chain messages. Bonded bridge validators that fails to delivery on their responsibility will be punished.

3.3 Delegators

A delegator is a stake-holding party who contributes to the security bond of a chain or bridge validator. They have no additional role except to place capital risk and as such to

signal that they trust a particular validator (or set thereof) to act responsibly in their maintenance of the network. They receive a portion of the transaction fees paid out to the validators proportional to their stake contribution.

4. System Design

The system can be roughly broken down into four components: the chain operating system, the consensus mechanism, the interchain transaction protocol and the application protocol.

4.1 Chain Operation

The blockchain system for app-chains will likely be a system broadly similar to Ethereum in that it is state-based with the state mapping address to account information, mainly balances and (to prevent replays) a transaction counter. A notable different is contracts cannot be deployed through transactions; following from the desire to avoid additional functionalities aside from the intended application logic, it will not support public deployment of contracts.

The chain's VM would be based around the EVM, it would have a number of modifications to ensure maximal simplicity while remain its turing-completeness to allow for application layer logics support. It would likely have a number of built-in contracts (similar to those at addresses 1-4 in Ethereum) to allow for platform-specific duties to be managed including a consensus contract, a validator contract and a permissioning contract.

The functions available for public usage will consume fixed amount of computational resources ("gas"), as such, a flat fee will apply in all cases. However, following the rational that different functionalities might have different gas consumption needs, and the desire to avoid coupling of logic between blockchain and application layer, the static implementation of gas will be enforced in the application layer.

4.2 Consensus

ContractLand achieves low-level consensus over a set of mutually agreed valid blocks through a modern asynchronous Byzantine fault tolerant (BFT) algorithm. The algorithm will be inspired by Parity Aura; the consensus mechanism of Parity's Proof-of-Authority (PoA) implementation [4]. PoA is a new family of BFT algorithms that achieves fault tolerance in a trusted set of validators. For consortium networks, this alone would be sufficient, however our vision of the application-specific blockchains are imagined to be also deployable as a network in a fully open and public situation without any particular organisation or trusted authority required to maintain it. As such we need a means of determining a set of validators and incentivising them to behave honestly. For this we utilise PoS based validator selection criteria.

4.3 Sealing Mechanism

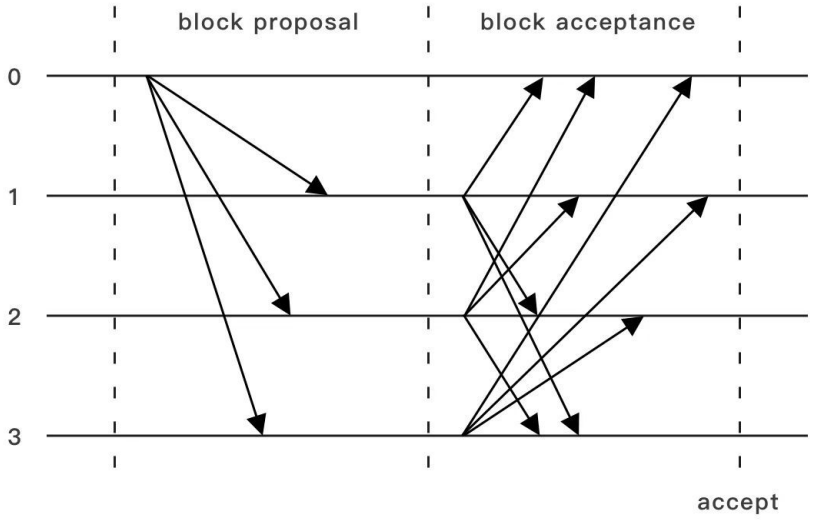

Figure 3. Message exchanges between validators to reach consensus. In this example there are 4 validators with id 0,1,2,3. The leader of the step is the validator 0.

The network is assumed to be synchronous and all validator nodes to be synchronised within the same UNIX time *t*. The index *s* of each step is deterministically computed by each

validator as *s = t/step duration*, where *step duration* is a constant determining the duration of a step. The leader of a step *s* is the validator identified by the id *l = s mod N*.

Validators maintain two queues locally, one for transactions *Qtxn* and one for pending blocks *Qb*. Each issued transaction is collected by validators in *Qtxn*. For each step, the leader l includes the transactions in *Qtxn* in a block *b*, and broadcasts it to the other validators (block proposal round in Figure 1). Then each validator sends the received block to the others (round block acceptance). If it turns out that all the validators received the same block *b*, they accept *b* by enqueuing it in *Qb*. Any received block sent by an validator not expected to be the current leader is rejected. The leader is always expected to send a block, if no transaction is available then an empty block has to be sent. If validators do not agree on the proposed block during the block acceptance, a voting is triggered to decide whether the current leader is malicious and then kick it out. An validator can vote the current leader malicious because (i) it has not proposed any block, (ii) it has proposed more blocks than expected, or (iii) it has proposed different blocks to different validators. The voting mechanism is realized through a smart contract, and a majority of votes is required to actually remove the current leader l from the set of legitimate validators. When this happens, all the blocks in *Qb* proposed by l are discarded. Note that leader misbehaviours can be caused by benign faults (e.g., network asynchrony, software crash) or Byzantine faults (e.g., the leader has been subverted and behaves maliciously on purpose).

4.4 Finality

Under the assumption of a synchronous network which propagates messages within the step duration *t*, let *SIG_SET(B)* be the set of signatures from all authors in the set of blocks *B*:

$$SIG\_SET(B)=\{a \mid \exists b \in B: AUTHOR(b)=a\}$$

If there is a valid chain *C* ending with *C[K..]*, where *|SIG_SET(C[K..])|>n/2, then C[K]* and all of its ancestors are finalized.

This definition of finality stems from a simple majority vote. In this setting, $2f+1 \leq n$, so the faulty nodes cannot finalize a block all on their own.

4.5 Validator Selection

Validator selection is done by staking tokens into a staking contract. A minimum staking amount of *S* is required to be considered a validator. All users with more than *S* deposited can become a bonded validator. Given there are *T* tokens in total supply, the max number of validators *N* at any given time in theory is $N <= T/S$. In practice, we expect at least 20% of the token supply will be free floating in the market for liquidity, meaning only at most 80% of *T* will be actively deposited as stakes for validators.

The staking contract maintains the validator set. It manages:

- Which accounts are currently validators.
- Which accounts are newly boned and will become validators in the next block.
- Which accounts have placed stake delegating to a validator.
- Staking volume of each validator.
- Benign and malicious behaviour proof verification and punishment logic.

It allows an account to register a desire to become a bonded validator (along with its requirements), to delegate to another account, for preexisting bonded validators to register their desire to exit this status, and verifies validator misbehaviour proofs and carries out the corresponding punishments.

4.6 Common Attacks

In this section we cover some common attacks PoS systems face and how they are addressed.

4.6.1 Nothing At Stake

The nothing at stake problem is a scenario where validators can effectively break safety by voting for multiple conflicting blocks at a given block height without incurring cost for doing so.

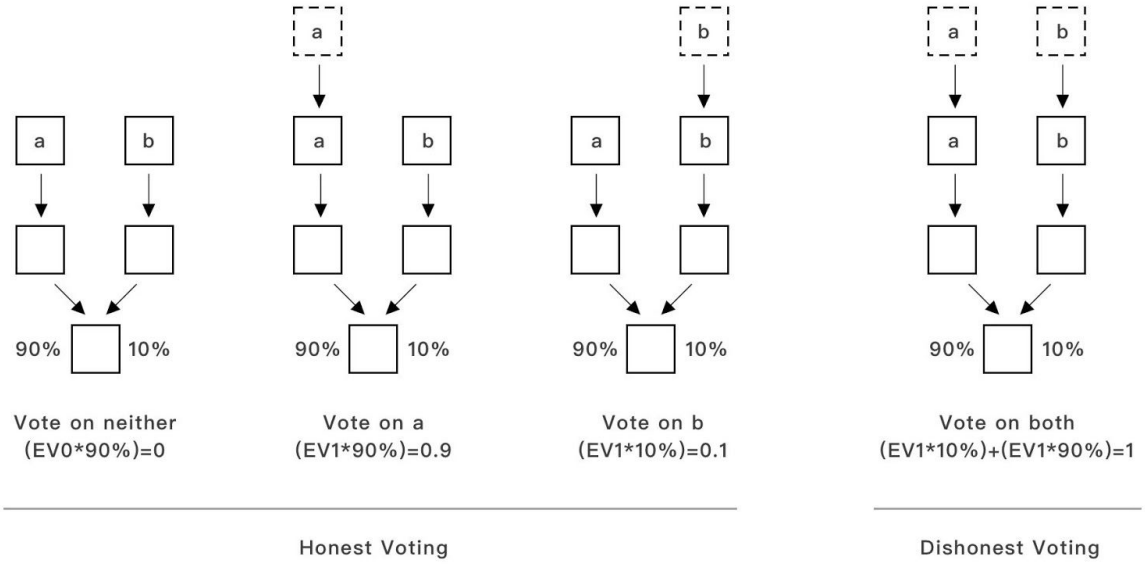

Figure 4. Expected gain for voting on competing chains is greater than expected gain for voting on a single chain in naive PoS design

Naive PoS implementations are vulnerable to these attacks. Since there's no incentive to ever converge (vote) on an unique chain, the optimal strategy economically becomes to vote on multiple conflicting chains at once to reap more block rewards. In Proof-of-Work, the "penalty" for mining on multiple chains is that a miner must split up their physical hashing power to do this, making it much more expensive and difficult to perform.

The original Aura implementation already contains mechanism to vote out validators for Byzantine behaviour, which includes proposing different blocks. An additional economical

punishment is added in the app-chain's consensus to further disincentivize this behaviour where the Byzantine validators' stake is taken away from possession by burning. This notion is broadly conceived a 'slasher', an idea popularized by Vitalik Buterin[5].

4.6.2 Long Range Attacks

In PoW blockchains, the longest chain is always the chain with the most hashing power. It would require a tremendous investment of computational energy to create a fork from some past block and fabricate valid blocks for it fast enough to catch up to the main chain.

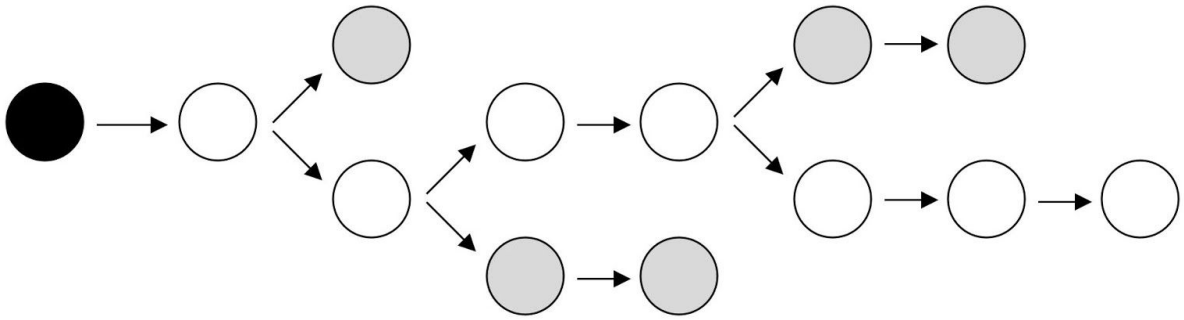

Figure 5. In PoW, the main chain is the branch with the greatest number of blocks, in this image, it is the chain with the black colored blocks

In Proof of Stake protocols, the longest chain rule is not enough to determine the main chain as validators in a PoS system can fabricate arbitrarily long chains very fast without incurring much cost. Unlike the short range nothing at stake problem, the burning of staking deposits does not prevent long range attacks as validators have the right to exist their validator role and withdraw their deposits. Once a malicious validator withdraws, it can build up a fork from an arbitrarily long range without fear of being slashed.

App-chain's block finality and sealing mechanism defends against long-range attacks naturally. The finality rule states that any chain *C* that has been approved by more than *N/2* validators is finalized, therefore any blocks that are proposed by the malicious leader that

is not compliant with the history of chain *C* is refused by other validators. The attacker will then be slashed and kicked out of the validator set.

4.6.3 DDoS

Distributed Denial-of-Service attack (DDoS) is a common form of attack in the field of cloud computing [6]. In the context of a blockchain network, it can be used to affect the proper functioning of nodes in attempt to escalate the attack surface. For example, a well executed DDoS attack could bring a good portion of the validator nodes offline, during which the attacker could gain a large enough portion of the validating power to take over the network. Validators as descripted in our framework are heavily bonded and could be targets of DDoS attacks.

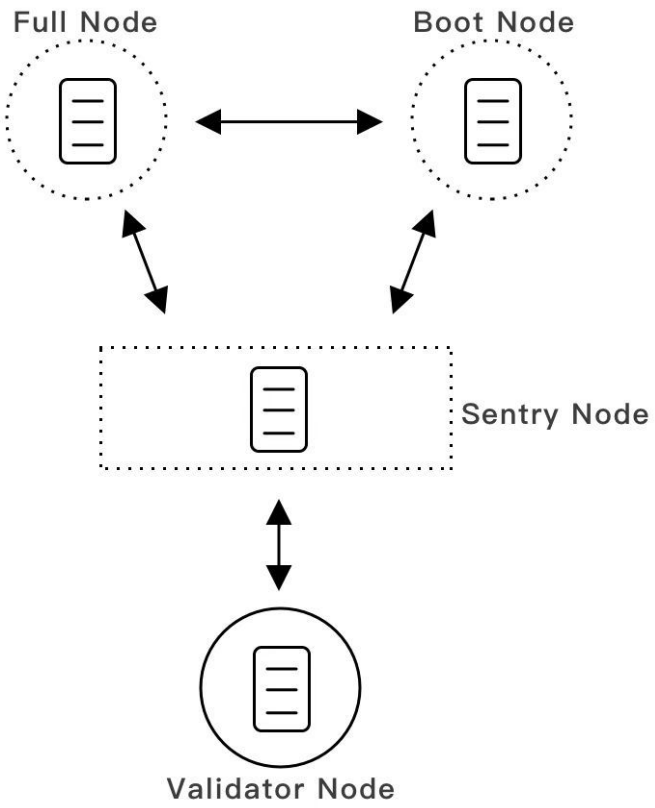

Figure 6. A sentry node sits in-front of a validator node, protecting it from public internet traffics

In additional to common DDoS prevention methods that are expected to be taken by validators, the framework offers an additional layer of protection with a dedicated type nodes called *Sentry Nodes*. Sentry Nodes is a concept following the sentry architectural pattern. They act as guardians for validator nodes and provide them with access to the rest of the network. They should be well connected to other full nodes on the network while the validator node could remain hidden from the public internet. Sentry Nodes may be dynamic, but should maintain persistent connections to some evolving random subset of each other. They should always expect to have direct incoming connections from the validator node and its backup(s).

4.7 Interchain Communication

A critical ingredient of the framework is interchain communication. The communication across chains is made via the concept of a *bridge*. The concept itself is simple: transactions executing in a connected chain are (according to the logic of that chain) able to effect the dispatch of a transaction into the app-chain. To ensure minimal implementation complexity, minimal risk and minimal upfront architectures, these interchain transactions are effectively indistinguishable from standard externally signed transactions.

4.7.1 Bridge Overview

The bridge is a two way pegging mechanism that works through two bridge contracts ("contract" here refers to an executable piece of logic in the context of its chain). The bridge contracts resides on the *Home* and *Foreign* chains. The concept of home and foreign is relative depending on the context. Speaking from the perspective of an app-chain, *Home* would refer to that chain itself, while *Foreign* refers to any other app-chains or public chains that the home chain is connected to through the bridge.

The bridge contract must be able to accept and lock funds, verifies cryptographic signature of incoming cross-chain transfer transactions, and release token to user address on

successful transfers. The relay of messages between the two bridges on different chains happen in a byzantine fault tolerant way by the Bridge Validators.

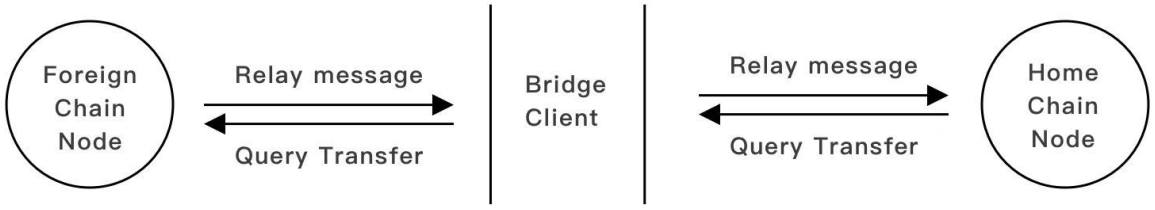

Figure 7. Communication flow between terra-chain and connect chains

The simple flow of how the transfer is performed between bridges and validators is:

1. User *U* deposits some amount *T* of coins *Cf* to the foreign bridge at address *Bf*. The transaction contains metadata for validators to relay the transfer:
    - *Cf* - coin address in foreign
    - *T* - transfer amount
    - *R* - recipient address
2. Validator queries and find a new incoming transaction on the address *Bf* on Foreign Chain.
3. Validators *(1..N)* sends message to the home bridge at address *Bh* to relay the transfer with the following parameters:
    - *Cf* - coin address in foreign
    - *R* - recipient address.
    - *T* - transfer amount
    - *TX* - hash of deposit transaction into *Bf*
    - *Sig* - validator signature of the message
4. The home bridge receives the incoming message, verifies its signature and keeps track of the signatures collected from validators. When more than *N/2* validator

signatures are collected for a given *TX* then *T* amount of home version of *Cf*, *Ch* is transferred from *Bh* to *R*.

4.7.2 Bridge for EVM Chains

Due to similarities of EVM based chains and ContractLand's app-chains, we expect there is ample opportunity for app-chains to be interoperable with any EVM based chains. The Foreign bridge can be implemented via smart contracts in Solidity. Using EVENTs, validators can efficiently be notified of incoming transfer requests from both side of the bridge. Messages relayed by the validators into the bridge contracts are signed with elliptic curve digital signature (ECDSA) and validated on-chain using *ecrecover* [7].

Through the choice of a BFT consensus mechanism with validators formed from a set of stakeholders determined by depositing stakes in the bridge staking contracts, we are able to get a secure consensus with an infrequently changing and modest number of validators. Finality of cross-chain transfer is achieved when *N/2 + 1* validators have confirmed and verified the transfer.

In this model, bridge validator nodes would have to do little other than listen for events, sign messages and send transactions to bridge contracts. To receive the events from and get transactions actually routed onto the foreign and home chains, we assume either validators themselves would also reside on the respective networks (i.e running their own full nodes) or, utilize public node services (such as Infura [8] for the Ethereum network). The latter while being a more lightweight method, involves a trust factor in the public node provider.

4.7.3 Bridge for Bitcoin Like Chains

The challenge with Bitcoin is how the deposits can be securely controlled from a rotating validator set. Unlike Ethereum which is able to make arbitrary decisions based upon combinations of signatures, Bitcoin is substantially more limited, with most clients

accepting only multisignature transactions with a maximum of 3 parties. Extending this to tens, or indeed thousands as might ultimately be desired, it is impossible under the current protocol. One option is to alter the Bitcoin protocol to enable such functionality, however so-called "hard forks" in the Bitcoin world are difficult to arrange judging by recent attempts. Another alternative is to use threshold signatures, cryptographic schemes to allow a singly identifiable public key to be effectively controlled by multiple secret "parts", some or all of which must be utilised to create a valid signature. Unfortunately, threshold signatures compatible with Bitcoin's ECDSA are computationally expensive to create and of polynomial complexity. Other schemes such a Schnorr signatures provide far lower costs, however the timeline on which they may be introduced into the Bitcoin protocol is uncertain.

Since the ultimate security of the deposits rests with a number of bonded validators, one other option is to reduce the multi-signature key-holders to only a heavily bonded subset of the total validators such that threshold signatures become feasible (or, at worst, Bitcoin's native multi-signature is possible). We can achieve this using a M-of-N P2SH multisignature address according to BIP-13 [9]. The Bitcoin reference implementation has validations rules limiting the P2SH redeem script to be at most 520 bytes. The redeem script is of the format:

[M pubkey1 pubkey2 ... N OP_CHECKMULTISIG]

It follows that the length of all public keys together plus the number of public keys must not be over 517 bytes. For compressed public keys, this means up to N=15.

This of course reduces the total amount of bonds that could be deducted in reparations should the validators behave illegally, however this is a graceful degradation, simply setting an upper limit of the amount of funds that can securely run between the two networks (or indeed, on the % losses should an attack from the validators succeed).

4.8 Application Layer

The application layer contains logics of the application domain for the blockchain in the form is smart contracts. From the chain operating system perspective, the high level contract code is compile down to EVM byte codes, and stored on-chain. Each full node of the application chain will have a copy of the application logic in the form of on-chain state data. Therefore no single party could modify the application logic. The framework by design does not impose any restrictions or rules on the application logics. Design decisions and implementation details (such as upgradability and governance) are open ended and up to the developers of the application-specific chains to determine.

4.8.1 Example Use Cases

Here we examine some example application scenarios that can benefit from building on the application-specific blockchain framework.

Next-gen decentralized exchange. One obvious utility of the framework is decentralized exchanges (DEX). Most DEXs existing today are hybrid exchanges, usually offloading the orderbook and matching engine off-chain. The end-user experiences are usually clunky due to public blockchain's performance limitations. A DEX built using the application-specific blockchain framework can have the entirety of the exchange system built on-chain, and expect end-user experience similar to centralized exchanges. Utilizing the bridge, the DEX would open up assets available for trading to all existing blockchain systems instead of being limited to only assets available on the chain the DEX was deployed under. A pure DEX built this way could provide stronger security guarantee, stronger censorship resistance, and minimize centralized operational risks.

Peer-to-peer gambling. Any number of peer-to-peer gambling protocols can be implemented using the framework including advanced games such as poker and mahjong with relatively intensive performance requirements. A wide selection of assets could be

used as part of the games with help of the bridge.

Gaming. Blockchain based games popularized by CryptoKitties infused blockchain's immutability properties in games. However, current blockchain based games are low in actual gaming experiences as they are mainly oriented on the collectable and token aspects. As performance and asset isolability constraints are lifted, more complexed gaming experiences could be created, opening up more possibilities for the decentralized gaming field.

5. Conclusion

We have outlined a general framework for creating scalable decentralized applications using application-specific blockchains and cross-chain communication bridges. As the system requires different roles in it's upkeep, we've included behavioral mechanism through proper incentive and penalties. We have touch upon in detail on the consensus mechanism improvements of the blockchain layer, the bridging protocol design and limitations of the protocol for different blockchain systems, and concluded with example application use cases for the framework.

6. References

[1] Delegated Proof-of-Stake consensus.
https://bitshares.org/technology/delegated-proof-of-stake-consensus.

[2] Go Ethereum (Geth). https://github.com/ethereum/go-ethereum.

[3] Solidity. https://solidity.readthedocs.io/en/v0.4.24.

[4] Proof-of-Authority consensus in Ethereum.
https://wiki.parity.io/Proof-of-Authority-Chains.

[5] Slasher: A Punitive Proof-of-Stake Algorithm.
https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm.

[6] DoS and DDoS Evolution. http://users.atw.hu/denialofservice/ch03lev1sec3.html.

[7] Solidity, Mathematical and Cryptographic Functions.
https://solidity.readthedocs.io/en/latest/units-and-global-variables.html#mathematical-and-cryptographic-functions.

[8] Infura, public Ethereum infrastructure provider. https://infura.io.

[9] BIP13, Address Format for pay-to-script-hash.
https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki.