ICTP
A highly scalable blockchain-based system for currency, computation, and contracts
DRAFT v3.6

Jessica Taylor Jack Gallagher

Baeo Maltinsky Kaavya Jayram Wei Dai

Contents

1 Introduction

Cryptocurrencies have seen substantial academic and speculative interest but comparatively little widespread adoption. While the theoretical basis of distributed ledger technology is promising, the technical limitations of current systems prevent them from meeting the requirements for mundane utilities: low throughput compromises system reliability,[1] and transactions lack the privacy users have come to expect from payment systems. The currencies built on these platforms are thus seldom used outside communities of blockchain enthusiasts, speculative traders, and individuals seeking to circumvent regulation.

At the time of writing, the most popular platforms cannot achieve sufficient scale to handle the economy of a single major metropolitan area, let alone the volume of transactions that comprise global trade.[2] Because the current generation of distributed ledger technology has only reached this immature, proof-of-concept stage, its ability to support novel infrastructure projects suited to such a platform remains doubtful. Projects such as micropayments, decentralized energy grids, and peer-to-peer financing networks rely on network effects to succeed, and existing blockchain tech has not demonstrated reliability at scale.

In this paper we present the Inductive Consensus Tree Protocol (ICTP), a new protocol for distributed consensus and computation at scale. It achieves high throughput, $\sim$ \$0.0002 transaction costs, and high security while preserving privacy.

The protocol's efficiency relies primarily on the *inductive consensus tree*, a novel use of the distributed radix hash tree[3], which spreads computation and validation out among stakers to minimize redundancy and requires only local constraint validation to produce globally valid data. The structure of the inductive consensus tree results in exponential dropoff in probability of fraud as parameters are tuned. Even with a pessimistic expectation of constant coordinated attack by 1/3 of staking nodes, the fraud rate is less than once in one trillion years. For privacy, we make use of zero-knowledge proofs [3], which allow auditing of all public and private transactions in the system without compromising the confidentiality of private state.

The design decentralizes verification, staking, and storage, and supports tens of millions of transactions per second, with capacity increasing as global bandwidth grows.

2 Summary of Results

We specify a blockchain system with the following properties:

- *Scalability*: The total amount of computation that must be done to verify and include a transaction is polylogarithmic[4] in the size of the network.

- *Decentralization*: Verification and indexing of data in the system is distributed among

[1]Best evidenced by the catastrophic interruption of the Ethereum network in late 2017 upon the release of CryptoKitties.

[2]The daytime population of Manhattan is roughly four million; Ethereum supports 1.3 million transactions per day, and Bitcoin only 605,000.

[3]also known as a Merkle [1]-Patricia [2] tree

[4]i.e., polynomials of the logarithm, not to be confused with the family of polylogarithm functions

many parties, none of whom are burdened by large amounts of computation. It is possible to participate as a staker in ICTP consensus using a low-end system such as a Raspberry Pi, or a free-tier AWS instance.

- *Security*: Under sustained coordinated attack by up to 1/3 of stakers, assuming secure cryptographic primitives, ICTP transactions are unforgeable and the ledger remains immutable. Attackers cannot insert invalid data into the system, as this data will be verified by a random subset of nodes, which is selected such that the chance that this group approves an invalid transaction is beneath 1 in 10^{24}.

- *Reliability*: Under the assumption that at least 60% of stakers are online and participating honestly in consensus, the entire system only experiences multi-minute downtime once every 800 trillion years.[5]

- *Privacy*: Account balances and transaction histories are private by default. Senders reveal neither the amount nor the identity of their receiver, while receivers need only know the sent amount, but not the identity of the sender. Transactions are still verifiable while preserving privacy using zero-knowledge proofs.[6]

- *Scriptability*: ICTP contains a Turing-complete account scripting system, built on WebAssembly, which shares the same scaling, security, and privacy properties that ordinary transactions do. Smart contracts[4] communicate with each other and with ordinary accounts using the actor model. Using zero-knowledge proofs, contracts can keep data private while allowing verification of correct execution. Thus, it is possible to build systems such as private peer-to-peer credit networks on ICTP.

3 Cryptographic Primitives

ICTP relies on the following cryptographic primitives:

- Public-key digital signatures: we plan to use 3072-bit RSA.[5][7]
- SNARKs (Succinct Non-interactive ARguments of Knowledge): we plan to use a protocol due to Groth.[6]
- A hash function: we plan to use MiMC[7], a hash function that is easy to evaluate in a SNARK.

[5] As often as once every 200,000 years, somewhere in the tree may experience downtime, but this is generally a trivial fraction of the overall system. See Section 13 for a detailed analysis.

[6] While full privacy requires obfuscation of network traffic (a subject beyond the scope of this paper), such obfuscation is straightforward (e.g., by routing network traffic through Tor) given that the protocol itself does not require anyone to know the private account data of anyone else.

[7] We use RSA instead of ECDSA because verification involves checking many signatures, and verifying an RSA signature is much faster than verifying an ECDSA signature.

4 Brief Overview

The following is a brief overview of ICTP. This is not a technical specification but will motivate and provide context for the definitions given later.

ICTP organizes a system of accounts, some of which are owned by a particular party (and correspond to a public key), and some of which are controlled by a smart contract. Accounts have both public and private states, which include public and private balances.

An *inductive consensus tree* (ICT) is a radix hash tree that contains, for all accounts, the public state and the hash code of the private state.[8] The data in the ICT (including nodes and public account states) is stored in a distributed hash table (*DHT*), which ensures it is accessible by everyone.

The main blockchain is proof-of-stake[8], and each block contains the hash code[9] of the current ICT. A new ICT is produced each block using an iterated consensus procedure. The new ICT will reuse many of the nodes of the previous ICT, ensuring that only a limited number of new nodes are created per account update.

Accounts can perform certain *actions* (e.g. sends and receives), which create new account states from previous states. Actions may be public or private. When an action modifies the private state of an account, a SNARK must be provided showing that the new hash code of the private state is in fact the hash code of a new valid state.

5 Data Structures

Here we provide a sketch of important data structures used in the system. These definitions are not complete and omit some fields from the data structures, but we provide enough detail that it is possible to understand the semantics of the system and of the consensus algorithm.

5.1 Accounts

Accounts are represented by a particular public key, and account IDs are the *hash code* of this public key.

5.2 Main blockchain

ICTP uses a proof-of-stake blockchain. Each main block contains:

- prev, the hash code of the previous main block (this is included for all non-genesis blocks).
- blockIndex, the index of this block according to the block order. The genesis block has blockIndex 0, the next block has blockIndex 1, and so on.

[8]Some intermediate ICT nodes contain signatures by a pool (see Section 6).

- timestamp, the time (in Epoch seconds) that the block was created, rounded to the nearest multiple of the standard block time (around one minute).
- tree, the hash code of the top node of the current ICT (see Section 5.3).
- Information relevant to generating pseudorandom numbers (see Section 9).
- Signatures of all the above by the signers and the miner.

5.3 Inductive Consensus Tree

ICTP uses an *inductive consensus tree* to store all public account information, hash codes of private account information, and other data relevant to verification. It represents the current state of the system.

More specifically, the inductive consensus tree is a radix hash tree. Nodes are either branch nodes or leaf nodes. A leaf node represents a particular account, and the path of that node is the ID of the account.

All ICT nodes contain:

- lastMain, the hash code of the previous main block.
- path, a list of characters indicating the path from the top node to this node.
- Metadata about this node, including the total amount of stake in the accounts under this node.

Branch nodes, a type of ICT node, have the following additional fields:

- children, a list of `(path suffix, hash code of child ICT node)` pairs, such that the path of each child is the path of the parent node concatenated with the corresponding path suffix. No two path suffixes may share the same first element.
- Endorsed branch nodes contain the hash code of a list of signatures of the node body. The signatures attest to the validity of the node body and must correspond to a large enough subset of pool members (pools and endorsement will be defined in Section 6).

Finally, leaf nodes, also a type of ICT node, contain:

- pubData, the hash code of the top node of the data tree (see Section 5.4) representing the public state of this account.
- privData, the *salted* hash code of the top node of the data tree representing the private state of this account.
- newSNARKs, a list of SNARKs that together prove that the new account state follows from the previous state.

Note that the children of an ICT node may include old nodes (i.e. one with an earlier lastMain); so only a *logarithmic* number of nodes need to be changed per account state update.

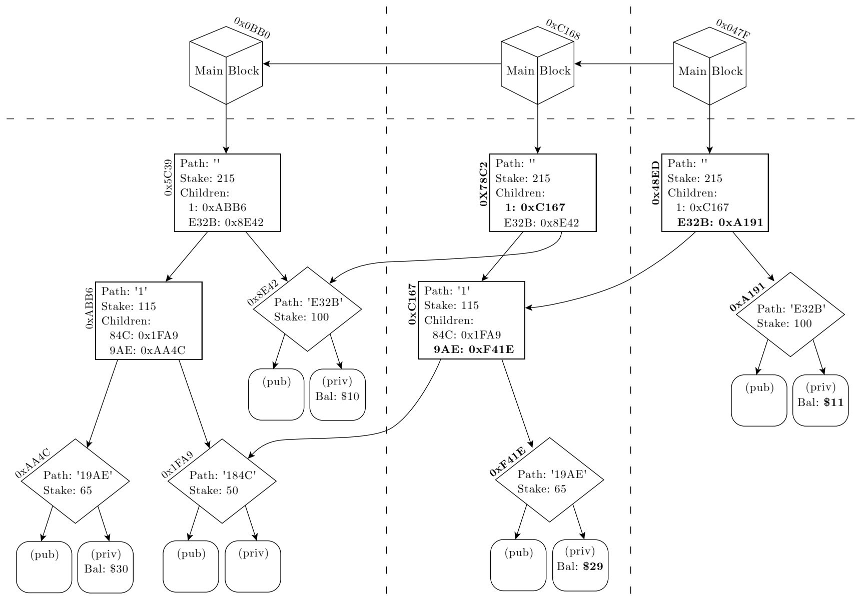

Figure 1: Sequence of simplified ICT updates demonstrating account `0x19AE` sending $1 to account `0xE32B`.

5.4 Data tree

Data trees store public and private account data. Like ICT nodes, data trees are radix hash trees. Semantically, they map strings (paths in the radix hash tree) to dictionaries representing field values.

Data nodes contain:

- fields, a dictionary mapping field names to field values. Field names are strings, and field values can be any binary-encodable data.

Branch nodes, a type of data node, additionally contain:

- children: This field serves the same purpose as the children field in an ICT branch node (see Section 5.3).

5.5 Actions

An action is a description of an operation modifying an account state. Some actions are private and some are public. All actions include the following:

- lastMain, the hash code of the last main block.
- type, the type of action.
- fee, the amount of money attached to this action that will be awarded to various parties involved in validating the action.

5.5.1 Send actions

A send action sends money and possibly other information. In addition to the universal action fields, a send action includes the following:

- recipient, the account ID of the message recipient.
- sendAmount, the amount of money to send.
- isPrivate, a Boolean indicating whether the send is from a public balance and is itself public, or is from a private balance and is itself private.
- isReceivePrivate, a Boolean indicating whether the receive corresponding to this send will be public or private, depending on the type of the recipient account.
- message, a dictionary describing additional data related to this send (which includes, for example, information about how contracts should interpret this message).
- salt, an arbitrary string determined randomly.
- signature, a signature of the action by this account.
- For send actions that initialize new accounts, information about this new account.

5.5.2 Receive actions

A receive action receives money from a send action. In addition to the universal action fields, a receive action includes the following:

- sendHash, the hash code of the send action to receive from.
- receiveAmount, the amount of money received (matching the sendAmount of the send action).
- message, a dictionary equal to the message of the send action.
- isPrivate, a Boolean indicating whether the receive is intended for a public balance and is itself public, or is intended for a private balance and is itself private.
- signature, a signature of the action by this account.
- For public receives, send, the actual send action whose hash is sendHash.

5.5.3 Other actions

Additional actions include:

- Claiming rewards.
- Punishing infractions.
- Moving money between private balance, public balance, and stake (which is always public).

5.6 Account states

Accounts have states, stored in the pubData and privData data trees. They include the following public data:

- publicKey, the public key corresponding to this account.
- stake, the amount that this account has staked.

Account states also include public and private versions of all the following data:

- balance, the amount of money that this account has in balance.
- A set of actions that have been taken on this account.
- A dictionary storing arbitrary write-once data, which may be modified by the account holder.
- A dictionary storing arbitrary multiple-write data, which also may be modified by the account holder).

An important aspect of this encoding is that any of these pieces of data can be accessed or modified with work that is on average logarithmic in the number of actions taken on this account.

5.7 Action semantics

Action semantics specify whether the new state of an account is valid based on the previous state of the system.

All actions must satisfy the following to be valid:

- lastMain must be the hash code of the previous main block.
- type must be the standard string representation of the type of the action (e.g., “send” for send actions).

- fee must be non-negative.

Certain types of actions have additional requirements for validity.

Properties of send actions

- recipient must be a well-formed account ID.
- sendAmount + fee should not exceed the previous account balance.
- The account must not have already sent the exact same send action (according to the action set).
- The action must not change the account state beyond two essential changes: reduction of the balance by sendAmount + fee and addition of the send action to the action set.
- The signature must be a valid signature by this account.
- If the send is private, a SNARK showing that the new private account state is valid must be included in the new leaf node.

Properties of receive actions

- A SNARK proving the following must be included in the new leaf node:
  - A send with hash sendHash exists in the previous ICT
  - The send has a sendAmount equal to the receive's receiveAmount
  - The send has message equal to the receive's message
- isPrivate must be equal to the isReceivePrivate of the send.
- For public receives, send must have hash code sendHash.
- The account must have not already received a send action with the same hash code (according to the set of actions).
- fee $\leq$ sendAmount + balance.
- The new account state is incremented by receiveAmount − fee, and the receive action is included in the updated action set. Apart from this, the account state remains unchanged.
- The signature must be a valid signature by this account.
- If the receive is private, an additional SNARK that shows that the new private account state is valid must be included in the new leaf node.

5.8 Distributed Hash Table

Most of the datan ICTP uses is addressed by the hash code of the data and stored in a DHT. Public data, such as ICT nodes and public data nodes, is put into the DHT so it can be accessed by anyone.

6 Concepts

6.1 Pools

A *pool* is a set of pseudorandomly chosen accounts that verify candidate ICT *node bodies*, the content of a node without the signatures it might gain during the verification process. After a pool is pseudorandomly formed, it operates for 10080 blocks before the pools are pseudorandomly formed again (See Section 9). Pool slots each have an index $i \in \mathbb{Z}_{|\delta|}$ that is used as an input to the random selection procedure.

Let $Q = \{\alpha, \mu, \Omega\}$ denote the set of pools where $\alpha \subset \mu \subset \Omega$. We define a function $\mathcal{T} : Q \to \mathbb{Z}^+$ where $\mathcal{T}(\delta)$ is the threshold number of pool member votes required to reach consensus and endorse a node body for a particular pool δ.

Associated with each pool is a smaller set of pseudorandomly chosen accounts called *coordinators* who are responsible for communication between different levels of the ICT and generating node bodies for endorsement by pools. We will denote the set of coordinators by C, where $C \subset \Omega$ and $|C| = 16$.

First, the coordinators send a node body to α for endorsement, where $|\alpha| = 126$. If this pool fails to reach consensus then they send the node body to μ, where $|\mu| = 170$. If this pool *also* fails, the process is repeated with Ω, where $|\Omega| = 1731$. When the requisite number of pool members sign a node, it is considered *endorsed*. The pool α requires $\mathcal{T}(\alpha) = 98$ signatures, μ requires $\mathcal{T}(\mu) = 123$ signatures, and Ω requires $\mathcal{T}(\Omega) = 866$ signatures. (See Appendix A for information on how these parameters were selected).

6.2 Validity

There are three nontrivial senses in which an ICT node can be correct:[9]

1. A node is *valid* if it is one of the following:
   - A leaf node whose account state logically follows from its previous account state.
   - A branch node whose updated state compared to its previous state satisfies the following.
     - No unmodified child nodes have been lost.
     - Any gained nodes are either:
       (a) Updated versions of its previous child nodes.
       (b) New endorsed child nodes.
2. A node is *endorsed* if it contains above a threshold number of pool member signatures attesting to the validity of the node (See subsection 6.1).
3. A node is *sound* if it and all its descendants are valid and endorsed, and all account states logically follow.

[9]Correctness also involves further technical constraints, e.g. well-formed instantiation of data types.

6.3 Incentives

To align the incentives of the participating accounts, ICTP uses prizes, staking rewards, and punishments. While these incentives are unnecessary to ensure the security of the system assuming an honest majority, they provide incentives for parties to remain honest.

- The *value* of an ICT node is determined by the transaction fees under the node.
- ICT nodes have *prizes*, payments taken out of the value of an ICT node and distributed among the verifiers who sign the node. The prize amount is specified in the ICT node. Accounts may claim prizes if they are pseudorandomly selected to receive them. Prizes are stochastic to reduce the number of prize claims that result from each pool.
- *Staking rewards* are given to miners and main block signers. They consist of the value of the top ICT node plus some amount of newly minted currency (determined by the inflation policy of the currency).
- *Punishments* are imposed on accounts who break certain rules, reducing the stake of these accounts.

Types of Punishments

- Punishments for miners and main block signers who sign multiple competing main blocks, and pool members who sign multiple competing nodes. These punishments remove the incentive to sign multiple main blocks or ICT nodes. Thus, signers are incentivized to sign the single block most likely to be part of the canonical chain, pool members are incentivized to sign the single ICT node most likely to make it into the main chain,[10] and miners are incentivized to sign only a single main block.
- Punishments for pool members who sign an ICT node that can be proven invalid. This reduces the incentive for pool members to commit fraud.
- Punishments for pool members who fail to include an *innocence proof* along with their signature; this innocence proof is a signature of the data they would have had to download in order to verify the proof. This reduces the incentive for a bandwagon effect where pool members sign ICT nodes that have already been verified without verifying the nodes themselves, since the proof requires the pool members to download all the data required to verify the node.

In general, miners and main block signers are incentivized to build on and sign only valid main blocks, since these are the ones most likely to be built on.[11]

[10]Which, due to the preponderance of honest pool members, must be valid (detailed in Section 12).

[11]Future honest miners and main block signers will check recent main blocks for validity before building on them.

7 Global Consensus Mechanism

New nodes are validated and included in subsequent ICT's through the following consensus mechanism.

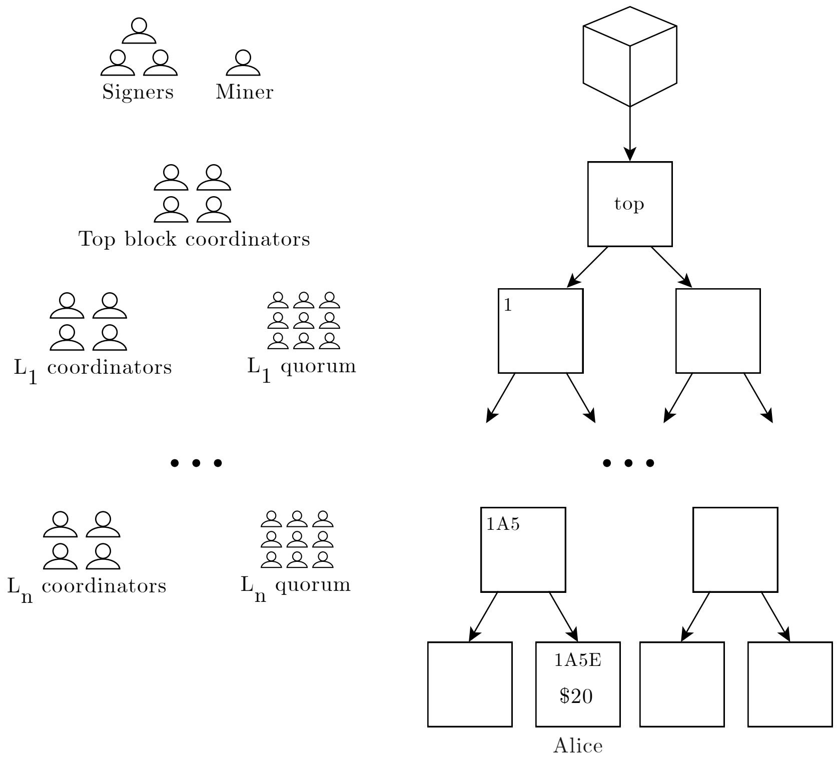

Figure 2: Illustration of the structure of an ICT.

Following an action causing updates to Alice's account state:

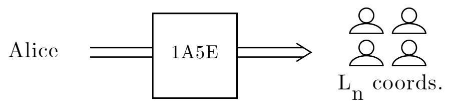

Figure 3: Consensus steps 1 and 2

1. Alice produces a SNARK proving that her new account state logically follows.
2. Alice sends her new leaf node, which includes this SNARK, to the coordinators of the lowest level of the ICT.

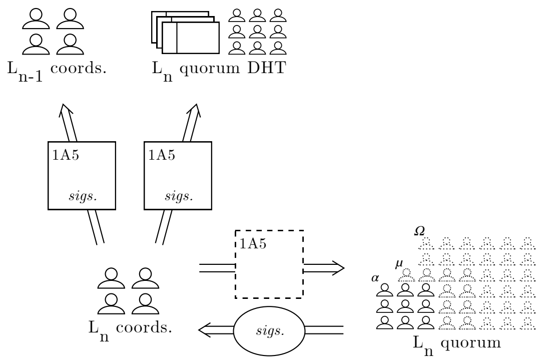

Figure 4: Consensus steps 3 - 7

3. The coordinators exchange data among themselves to ensure they all received the same inputs.
4. The coordinators each form the ICT node body and send this to the pool members.
5. The pool members check validity, endorse the node body and send it back to the coordinators along with innocence.
6. The coordinators put the endorsed node in the DHT.
7. The coordinators send the node to the coordinators of the next level above.

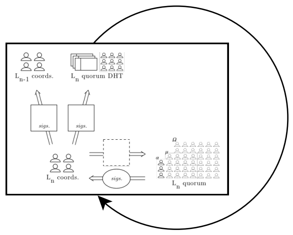

Figure 5: Consensus step 8

8. Steps 3-7 are repeated until the top level is reached.

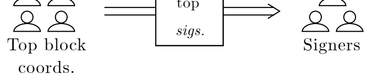

Figure 6: Consensus step 9

9. The coordinators of the top level send the top ICT node to the signers.

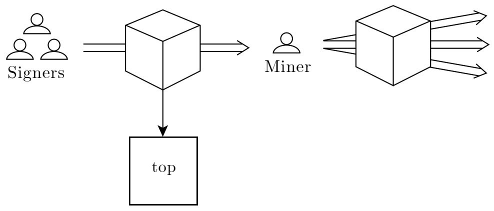

Figure 7: Consensus steps 10 and 11

10. The signers form the new block and send it to the miner.
11. The miner signs the top block and seeds.

8 Detailed Transaction Walkthrough

We now have enough data structure and concept definitions to describe how the ICT changes as Alice sends money to Bob.

We will assume that the send and receive are both private; the way public sends and receives work is slightly simpler. The process for Alice to send money to Bob is as follows:

1. Alice forms the following send action:

$$\begin{aligned}\text{Action} = \{&\mathsf{recipient} = \text{Bob's account ID},\\ &\mathsf{lastMain} = \text{hash code of last main block}\\ &\mathsf{type} = \text{send}\\ &\mathsf{fee} = \text{fee amount}\\ &\mathsf{sendAmount} = \text{sent amount},\\ &\mathsf{isReceivePrivate} = \text{true},\\ &\mathsf{message} = \{\}\\ &\mathsf{salt} = \text{random 32 byte string}\}\end{aligned}$$

2. Alice computes her new private state, which is the same as her old private state except that her private balance has been reduced by $\mathsf{sendAmount} + \mathsf{fee}$, and the new send action has been added to her private action set.
3. Alice computes the hash code of her new private state and creates a SNARK showing that her new private state is valid.
4. Alice forms her new leaf node, which contains her old public state (since her public state has not changed), the hash code of her new private state, and the SNARK she computed.
5. Using the consensus procedure, Alice gets the new leaf ICT node to be included in the next ICT; this process involves Alice sending the new leaf node to one or more pool coordinators, and many pool members validating the SNARK.

6. After the new ICT is finalized and included in the next main block, Alice forms a SNARK showing that there exists a send in the new ICT with the salted hash of the send action.

7. Alice sends both a salted hash of the send action and this SNARK to Bob.

8. Bob forms a receive action with lastMain set to the hash code of the last main block (which now includes Alice's send action), type set to "receive", fee set to the fee for the receive, sendHash set to the hash code of Alice's send action, and isPrivate set to true; he attaches his signature of these fields to the action.

9. Bob computes his new private state, which is the same as his old private state except that his private balance has been increased by the send amount minus the fee of the receive, and the new receive action is added to the private action set.

10. Bob computes the hash code of his new private state and creates a SNARK showing that his new private state is valid.

11. Bob forms his new leaf ICT node which contains his old public state, the hash code of his new private state, the SNARK Alice sent him, and the SNARK he computed.

12. Bob sends the new leaf node to one or more pool coordinators, and pool members validate both SNARKs by the usual consensus procedure.

At this point, excepting fees, Alice's balance has been reduced by the send amount and Bob's balance has been increased by this same amount. The process is complete.

9 Pseudorandomness

Pseudorandom numbers are used to select pool members, miners, and signers, and to award prizes.

ICTP generates an unpredictable random seed every 10080 blocks. Distributed random number generation has been considered difficult due to the Last-Actor Problem, in which the last participant to reveal their contribution to the process can gain substantial control over the output. ICTP addresses this using a distributed random number generation algorithm based on threshold cryptography[10] similar to those proposed by Hanke,[11] Drake,[12] and Robinson.[13]

We begin with an overview of the procedure, with a more detailed description presented below. There are a number of optimizations and tweaks that could be incorporated to improve the efficiency of this procedure which are beyond the scope of this paper.

Overview

A group of participants generate random polynomials of a certain degree, evaluate them on a number of distinct values (equal to the number of other group members), and then make a single point available to each other member of the group. All honest members then publish the points they receive. The degree of the polynomial is set to be substantially lower than the number of evaluations, so as long as a certain *threshold* of group members are acting honestly, the honest members will be able to recover the polynomials and combine the coefficients to produce a random seed.

Detailed Description

The generation of the random seed will be performed by a special, single-purpose pool γ. Let $g = |\gamma| = 701$.[12] The algorithm proceeds as follows:

1. All members of γ generate an ephemeral public and secret key pair (ephemeral because the secret keys will be revealed later in the procedure) and exchange public keys with all other members.

2. Let the threshold $t = \lceil \frac{g}{2} \rceil = 351$. Each member of γ generates a random polynomial of order $t-1$ over the prime field $\mathbb{Z}_t$. To be specific, each member $\gamma_i \in \gamma$ produces a set of random coefficients $p_{i,0}, p_{i,1}, \ldots, p_{i,t-1} \in \mathbb{Z}_{701}$ which represent the polynomial
$$p_i = p_{i,1}X + p_{i,2}X^2 + \ldots + p_{i,t-1}X^{t-1} \mod g$$

3. For each member $\gamma_i \in \gamma$, γ_i evaluates their polynomial p_i on each point $j \in \mathbb{Z}_{701} \backslash \{i\}$.

4. Each member $\gamma_i \in \gamma$ commits to the root of the hash-tree representation of p_i. Members who fail to commit after a certain period has passed will be ignored by honest members for the rest of the process.

5. Each member $\gamma_i \in \gamma$ encrypts $p_i(j)$ with γ_j's public key for each $j \in \mathbb{Z}_{701} \backslash \{i\}$ and publish the resulting ciphertexts. Members who do not post all $g-1$ encrypted values after a certain period of time will be ignored.

6. When the previous phase is complete, all members share all plaintext points they have received.

7. Upon observing 351 points of p_i, each member interpolates on those points to obtain the original polynomial and reconstructs its hash-tree representation, checking that the points determine a polynomial of degree $t-1$ with the originally published root.

8. The hash-roots of the recovered polynomials are XOR'd together and hashed again to produce a new random seed.

This process is robust and only requires t members to be acting honestly in order to succeed. Assuming $b = 1/3$ of stake is held by coordinated malicious actors, the attackers will have control over the random seed at a rate of one in 2.02×10^{17} shuffles, or once in 386 quadrillion (10^{15}) years. If $h = 6/10$ of stake is held by honest parties, the procedure will fail to produce a random seed at a rate of one in 22.9 million shuffles, or once in 439 thousand years.

The state of the random number generator is stored in the main chain. For each message, there are many (~ 64) consecutive blocks in which it may be validly included, so these messages will get included in the main chain with very high probability ($> 1 - 2^{-64}$) if most miners/signers are honest.

[12]Note: g is prime to make the finite field arithmetic simpler.

Selecting a Pseudorandom Account According to Stake

At multiple points in the validation process, it is necessary for some node to randomly select an account weighted by its stake. These accounts are selected using pseudorandom numbers and a recursive descent procedure. In general, stakes at a time *before* the pseudorandom number generation process starts are used as weights; this is to prevent attackers from moving stake around based on information they have about what random numbers will be generated.

Given a new pseudorandom seed, how can we randomly sample an account by its stake? We will describe a general procedure that, given an ICT node, randomly selects an account under that node, weighted by stake. For leaf nodes, the process is trivial; simply return the account corresponding to the leaf node. For branch nodes, the procedure is to randomly select an immediate child node (which may be a leaf node or a branch node) weighted by the stake in this node, then recursively apply the procedure to the child. To help in efficiently randomly selecting a child, each ICT node contains, as part of its metadata, the total amount of stake in this node (i.e. in accounts at or under this node). In total, the computational work and bandwidth required to randomly select a stake-weighted account is linear in the depth of the tree, and therefore logarithmic in the number of accounts.

It is easy for miners and main block signers to tell that they are miners or signers, since there are only a limited number of these per block. However, due to the size of the ICT, pool members will have difficulty telling which pool(s) they are part of. To ensure that all members know which pools they will belong to, we will use a recursive procedure that can be thought of as an *information waterfall*. For a given pool, each member of said pool will inform every member of the child pool. Assuming at least half of the large pool is honest, this procedure will theoretically succeed with probability $\geq 1 - 2^{-\Omega}$.

10 Smart Contracts

Smart contracts on ICTP are accounts with extra fields for contract code.

A contract state consists of:

- Normal account data without public key or stake.
- The hash of contract code.
- The hash of initialization parameters.

All interactions with contracts are through commands. Commands consist of a command string, a list of arguments, and a list of send actions that the command is receiving. If the list of send actions being received is empty, a fee is taken out of the contract balance.

Contracts are specified in WebAssembly.[14] Each contract must include code specifying three functions:

- `init`, which is called on contract initialization
- `command`, which produces a new state.

- query, which produces a result but does not modify the state.

Additionally, contract module code is run through a preprocessor that performs the following modifications:

- inserts a step-counter which computes gas costs during code execution
- canonicalizes nondeterministic outputs such as floating point numbers

There will additionally be API calls for the following features:

- Sending and receiving money.
- Reading and writing data fields
- Verifying SNARKs for the following circuits:
  - Proving the existence of a data node (given by its salted hash) at a location (also specified by salted hash) in the ICTP data tree or under a given Merkle root.
  - A Turing-complete vCPU circuit.

These features allow for creation of contracts that have all the privacy features of the base system, including for example private tokens built on top of ICTP, or contracts operating with entirely private code on top of the vCPU circuit. In principle, ordinary accounts on ICTP could themselves be implemented by a standard contract with commands for sending, receiving, staking, and other actions. Although this paper presents ordinary accounts and contract accounts separately, we plan to unify them in a later specification.

11 Scalability

11.1 Number of new nodes created per transaction

Let N be a set of account IDs, each a random 256-bit string, where $|N| = n$. Let m denote the number of transactions per block, each performed by the account associated with a random element of N. Each transaction creates exactly one leaf node, and some number of branch nodes. We will bound the total number of nodes created by a transaction.

Let $L_m = \lfloor \log_{16}(m) \rfloor$. We can bound the number of branch nodes created at or before level L_m by all transactions as:

$$\sum_{i=0}^{L_m} 16^i = \frac{1 - 16^{L_m}}{1 - 16} \leq \frac{16^{(L_m+1)}}{15} \leq m \cdot \frac{16}{15}$$

Now let us consider levels $L_m + 1$ and deeper.

Let A be a random element of N. Let X be the length of the longest prefix A shares with any of the other accounts (this corresponds to the length of the path from the root to the first common ancestor).

The probability that A shares the first k digits with another randomly chosen element $B \in N$ is 16^{-k}, since the accounts are chosen independently. By the union bound, the probability that it shares the first k digits with some other account is at most $16^{-k}(n-1) \leq 16^{-k}n$.

Let $L_n = \lfloor \log_{16}(n) \rfloor$. Now let us compute the expectation:

$$\begin{aligned}
\mathbb{E}[\max\{0, X - L_m\}] &\leq \sum_{x > L_m} P(X \geq x) \\
&= \sum_{L_m < x \leq L_n} P(X \geq x) + \sum_{x > L_n} P(X \geq x)
\end{aligned}$$

The probabilities in the first term can be upper-bounded by 1 (by definition). For the probabilities in the second term, we can use the union bound we derived above, so:

$$\begin{aligned}
\mathbb{E}[\max\{0, X - L_m\}] &\leq \sum_{x=L_m+1}^{L_n} 1 + \sum_{x=L_n+1}^{\infty} 16^{-x} n \\
&\leq L_n - L_m + \sum_{x=L_n+1}^{\infty} 16^{-x} n \\
&= L_n - L_m + \frac{n}{16 \cdot 16^{L_n}} \cdot \frac{16}{15} \\
&= L_n - L_m + \frac{n}{15 \cdot 16^{L_n}} \\
&\leq \log_{16}(n/m) + 1 + \frac{16}{15}
\end{aligned}$$

This bound on $\mathbb{E}[\max\{0, X - L_m\}]$ for A yields the expected number of branch nodes at level $L_m + 1$ or deeper created by a randomly chosen transaction performed by B.

So the total expected number of nodes created per block is at most:

$$\begin{aligned}
m \cdot \frac{16}{15} + m \cdot \left(\log_{16}(n/m) + 1 + \frac{16}{15} \right) + m &\leq m \cdot \left(2 + 2 \cdot \frac{16}{15} + \log_{16}(n/m) \right) \\
&\leq m \cdot (4.2 + \log_{16}(n/m))
\end{aligned}$$

which is $4.2 + \log_{16}(n/m)$ per transaction, showing that the number of nodes created per transaction scales logarithmically as the number of accounts in the system increases.[13]

We can construct a tighter bound by reasoning about the effects of a single transaction. Say Alice performs a transaction. She needs to create one new node (the root of her account state). A new node must then be created at the level immediately above her account state,

[13] It is worth noting that a higher frequency of transactions per block will increase the efficiency of the network as $\lim_{m \to n} \left(4.2 + \log_{16}\left(\frac{n}{m}\right)\right) = 4.2$.

but because some proportion of Alice's siblings also transact during this block, in expectation less than one node needs to be created as a direct consequence of Alice's transaction. As we move towards the root of the tree, the node produced at each level is distributed among a geometrically increasing number of other transactions. Specifically, the expected number of new nodes is $1/(1+(16-1)\frac{m}{n})$ at the lowest level, $1/(1+16(16-1)\frac{m}{n})$ at the second lowest level, $1/(1+16^2(16-1)\frac{m}{n})$ at the third lowest level, and so on (see Figure 8).

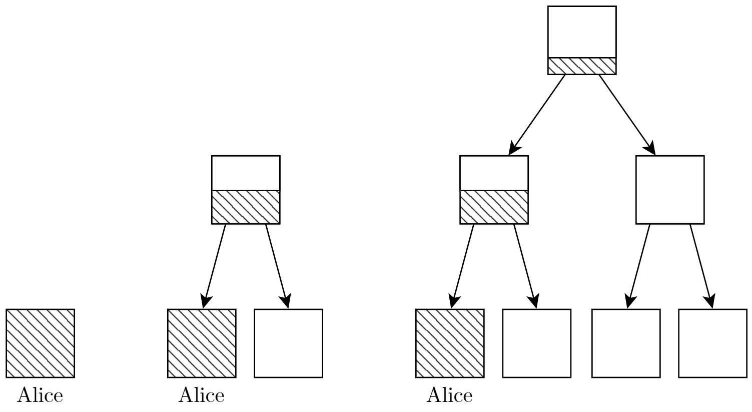

Figure 8: A simplified illustration of Alice's causal responsibility for newly created nodes.

Thus, the expected number of nodes created per transaction can be bounded by:

$$1+\sum_{i=0}^{\lceil \log_{16}(n) \rceil} \frac{1}{1+16^i \cdot 15\frac{m}{n}}$$

Examining the second term in the previous expression, we find that:

$$\begin{aligned}&\sum_{i=0}^{\lceil \log_{16}(n) \rceil} \frac{1}{1+16^i \cdot 15\frac{m}{n}} \\ &\leq 2+\log_{16}(n)+\frac{1}{\ln(16)}\left[\psi_q\left(\log_{16}\frac{-15m}{n}\right)-\psi_q\left(\log_{16}(-15m)+2\right)\right]\end{aligned}$$

where $q=16$ and ψ_q is the q-digamma function.[14] [15]

If we assume that the number of accounts in the system is $m = 10^5$ and each account transacts a bit over five times per year (a reasonable assumption if users establish hundreds of infrequently used contracts), then $n = 10^{10}$ and the expected number of nodes created by a marginal transaction is ~ 4.68.

[14]An analytic upper-bound for this form using elementary functions is left as an exercise to the reader.

11.2 Bandwidth per Transaction

Let h_{high} and h_{low} denote the optimistic and pessimistic proportions of stake on ICTP which are controlled by honest actors respectively, and let b denote the proportion of stake controlled by coordinated malicious actors. Under standard assumptions, this means $h_{\text{high}} = 0.8$, $h_{\text{low}} = 0.6$, and $b = \frac{1}{3}$.

The size of a SNARK is assumed to be upper bounded by 1 Kibit. Node bodies are assumed to be upper-bounded by 1 KiB. The signature size is 384 bytes. We will assume that a pool is capable of handling 1000 transactions per block. For the purposes of the following analysis, we will treat a send and its corresponding receive as one unit, and denote the number of such units a pool can handle by $t_b = 1000/2 = 500$.

The branching factor of the ICT is 16. Pools are reshuffled every 10080 blocks. During the reshuffling, pool members need to locate and obtain the RSA public keys (384 bytes[15]) of other pool members by descending the ICT (see Section 9 for details) and inform them of their pool membership. The cost of descending a single level is less than 1 KiB.

Let δ be a random variable sampled over the probability of using a particular element of Q (see Section 6). The expected pool size $\mathbb{E}[|\delta|]$ is a weighted average that can be calculated as:

$$\begin{aligned}
\mathbb{E}[|\delta|] =& |\alpha| + \Pr(\alpha \text{ fails to reach consensus}) \\
&\cdot \Big(|\mu| - |\alpha| + \Pr(\mu \text{ fails to reach consensus}) \left(|\Omega| - |\mu| \right) \Big) \\
=& |\alpha| + \left(\sum_{i=0}^{\mathcal{T}(\alpha)-1} \binom{|\alpha|}{i} h_{\text{high}}^{i} (1 - h_{\text{high}})^{|\alpha|-i} \right) \\
&\cdot \left(|\mu| - |\alpha| + \left[\sum_{j=0}^{\mathcal{T}(\mu)-1} \binom{|\mu|}{j} h_{\text{high}}^{j} (1 - h_{\text{high}})^{|\mu|-j} \right] \left(|\Omega| - |\mu| \right) \right)
\end{aligned}$$

The expected amortized bandwidth cost per transaction in KiB is given by:

[15]This number is potentially a few bits larger if the protocol does not specify a public exponent.

$$\begin{aligned}
\text{Bandwidth} =& \text{Lowest level} + \text{Alice to Bob} + \text{Other levels} + \text{Tree shuffle} \\
=& \mathbb{E}[|\delta|] \cdot 3 \cdot |\text{SNARK}| \\
&+ |\text{SNARK}| \\
&+ \left[\frac{|C|}{t_b} \cdot \mathbb{E}[|\delta|] \cdot (|\text{node body}| + 2 \cdot |\text{signature}|) \right. \\
&+ \frac{|C|^2}{t_b} \cdot (|\text{node body}| + |\text{signature}| \cdot \mathbb{E}[|\delta|]) \\
&\left. + 2 \cdot \mathbb{E}[|\delta|] \cdot \mathbb{E}[\#\text{ nodes created per txn}] \cdot |\text{node body}| \right] \\
&+ \left[\frac{|\Omega|}{\text{shuffle time} \cdot t_b} \cdot (\text{branching factor} + 1) \cdot |\Omega| \cdot (\text{descent cost} + |\text{key}|) \right. \\
&\left. + \frac{\lceil |\Omega|/50 \rceil \cdot |\Omega| \cdot \text{descent cost}}{\text{shuffle time} \cdot t_b} \right]
\end{aligned}$$

Given our assumptions, this comes out to about 1.52 MiB per transaction.[16]

11.3 Storage per Transaction

To ensure that the chance of obtaining a null pointer is less than 10^{-22}, the data will be replicated $R = \lceil \log_{(1-h_{\text{low}})} 10^{-22} \rceil = 56$ times.

Each transaction is at most responsible for work done by one pool, 1/16 of the pool above it, $1/16^2$ of the pool above that, and so on and so forth. This can be upper-bounded with the geometric series $\sum_{i=0}^{\infty} 16^{-i} = 16/15$. This cost is then amortized over the number of transactions per pool-block, t_b. This data will be stored for around one week. Additionally, the full path to an account state will be stored for a full year (52 weeks) to prevent loss of state.

The expected storage (in KiB-weeks) is then upper-bounded by:

$$\begin{aligned}
\text{Storage} =& R(\text{Short-term lowest level} + \text{Short-term other levels} + 52 \times \text{Long-term storage}) \\
=& R\Big[(2 \cdot |\text{node body}| \cdot \mathbb{E}[\#\text{ nodes created per txn}] + 3 \cdot |\text{SNARK}|) \\
&+ \big(|\text{node body}| + |\text{signature}| \cdot \mathbb{E}[|\delta|] \big) \cdot \frac{16}{15} \cdot \frac{1}{t_b} \\
&+ (52 \cdot \text{descent cost}) \Big]
\end{aligned}$$

Assuming 80% of the stake is controlled by honest accounts, the expected responsibility for storage per transaction is about 26 MiB-weeks.

[16]This calculation currently does not include bandwidth usage from packet framing or overhead from DHT routing, which likely increase the bandwidth usage by at least a few percent.

11.4 Current and projected costs

Currently, data transfer on AWS costs 6.5 cents per GB and storage on AWS costs 2.1 cents per GB-month. Using the derivations for bandwidth and storage in the previous two sections, the cost per transaction at launch is approximately 0.02 cents. The cost of a transaction on ICTP in 2029 is expected to be 0.0001 cents (see Appendix B.1).

11.5 Current and projected capacity

The number of transactions ICTP can support is limited primarily by global data transfer capacity, which can be conservatively estimated using global internet traffic (see Appendix B.2 for details).

We need to determine what percentage of global bandwidth will be used by ICTP to estimate its capacity. In 2015, Netflix made up 37% of internet traffic[16] and had a market cap of around 50 billion US dollars, which is the market cap of Bitcoin at the time of writing.

Assuming ICTP comprises 42% of global internet traffic, it will have a capacity of over 30 million transactions per second at launch and a projected 2029 capacity of over one billion transactions per second.[17]

12 Security

Fraud can occur if a pool $\delta \in Q$ is formed with a quantity of organized malicious actors exceeding $\mathcal{T}(\delta)$. We can pessimistically bound the probability of fraud by assuming the proportion of the stake controlled by organized malicious actors is $b = \frac{1}{3}$ and taking the union bound of the probability of fraud occurring in any given pool. This can be expressed as:

$$\sum_{\delta \in Q} \left[\sum_{i=\mathcal{T}(\delta)}^{|\delta|} \binom{|\delta|}{i} b^i (1-b)^{|\delta|-i} \right]$$

With our chosen parameters, this results in a fraud rate of less than 1 in 4.3×10^{24} per pool.

If the depth of the ICT[18] is 9, then there are no more than $16^8 \cdot \frac{16}{15}$ pools in the ICT. Thus, we can expect fraud to occur no more than once in every 1.84 trillion years. This is around 130 times the current age of the universe and close to the theorized lifespan of the longest lived stars.

In expectation, when the last dying stars burn out, as the sky goes dark and the cosmos grows cold, ICTP may not yet have seen a single instance of fraud.

[17] Again, we stress that these figures include both the send and the receive transaction.

[18] We place an explicit limit on maximum depth at which a pool may be formed, to ensure that attackers cannot search for an arbitrary location in the ICT that they control and then form a pool at that location.

If we assume that the system is only under constant organized attack by owners of one fifth of the stake rather than one third, the fraud rate drops to 1 in 10^{45} pools, or roughly once in 10^{33} years. This time-span is several orders of magnitude larger than shorter estimates for the half-life of proton decay,[17] implying that the universe may largely be composed of degenerate matter before the first instance of fraud on ICTP.

Other possible attacks on ICTP not already considered in this paper include:

- *Breach of cryptographic primitives*: The security of ICTP depends on the security of its cryptographic primitives (public-key cryptography, hash functions, and SNARKs). In particular, significant advances in quantum computing could make it easy to break the algorithms we use for public-key cryptography and SNARKs. Luckily, there exist alternative algorithms[18] [19] that cryptographers expect to be quantum-resistant, and more will be developed. If necessary, ICTP will switch to these primitives.
- *Denial of service*: Network nodes may be spammed with invalid or extraneous data, making it hard for them to process legitimate data. We plan to handle this type of attack using a combination of reputation systems and other standard methods to prevent denial of service.

13 Reliability

13.1 Consequences of pool Failure to Reach Consensus

Pessimistically assuming that the proportion of the staking pool controlled by honest actors is $h_{\text{low}} = 0.6$, Ω fails to reach consensus roughly once in every 440,000 trillion pools (one in 10-100 times the estimated number of ants on Earth[20]). With the same assumptions regarding the size of the ICT we used in Section 12, we can expect Ω will fail to reach consensus no more than about once in 200,000 years (roughly the age of the oldest uncontroversial evidence of modern *Homo sapiens* in Africa[21]).

In the event that Ω does fail to reach consensus, it is overwhelmingly likely to affect only a small proportion of the network. Specifically, if the ICT is $n+1$ deep then the proportion of accounts affected is given by,

$$\left(\sum_{i=0}^{n} 16^i\right)^{-1} \sum_{j=0}^{n} 16^j \cdot \frac{16^{n-j}}{16^n} = n\left(\sum_{i=0}^{n} 16^i\right)^{-1}$$

If $n = 8$, this is about 1.75×10^{-9}, or about 1 in 573 million accounts.

The result of Ω failing to reach consensus is not catastrophic, but it does potentially result in an account becoming inaccessible for the duration of the current ICT. Assuming a block time of one minute and a shuffling period of 10080 blocks, this implies an amortized average delay per account of less than one nanosecond per year.

13.2 Consequences of No Honest Coordinators

If it is possible that in a given block there will be no honest coordinators. The probability of this occurring, assuming the honest proportion of the staking pool is $h_{\text{high}} = 0.8$ is given by:

$$(1 - h_{\text{high}})^{|C|} \approx 6.6 \times 10^{-12}$$

or roughly one in every 150 billion pool-blocks. This produces an expected average delay per account of just over a microsecond per year.

14 Future Work

Features and improvements of ICTP that we are currently working on include:

- A distributed DNS system to support dynamic IPs. This would enable IP filtering and further ameliorate the effects of a denial-of-service attack.
- Using recursive certificates to add cryptographic rather than statistical guarantees against fraud.
- Formal verification of keys parts of the system.
- Looking into replacing RSA-3072 with a quantum secure cryptographic scheme based on lattices such as FALCON.[22]
- Investigating the possibility of replacing the linked list structure of the main blockchain with a skip-list.
- A more detailed analysis of the computational complexity of the network.
- Early confirmation mechanism.
- Modifying the branching factor of the ICT.

15 Acknowledgements

Thanks to Paul Christiano for AI safety ideas that inspired an earlier version of the protocol, Colleen McKenzie for designing the diagrams used in this paper, and Michael Vassar for organizing the trip that led to the inception of the project.

A Optimization

The sizes and consensus thresholds for pools were chosen with a re-sampling memetic inheritance algorithm[23] that minimized the amortized bandwidth cost per transaction while maintaining sufficiently low analytically derived upper-bounds for probabilities of consensus failure and fraud.

Let p be a probability we wish to minimize and t be the value we wish to target. Then we can construct the following penalty function:

$$\mathcal{P}(t,p) = \frac{10^6}{1 + e^{50((-\log_{10}(p)-t)-1)}}.$$

This penalty function grows exponentially when the difference between t and $\log_{10}(p)$ is too large; hence, minimizing $\mathcal{P}(t,p) + \text{Bandwidth}$ ensures that bandwidth is as small as possible while meeting security and reliability constraints.

B Cost and Capacity Projections

Historical trends in the costs of data transfer and storage can be well-described as decaying exponential functions, while global internet traffic has historically grown exponentially.

To forecast future trends, we performed weighted linear regression on the logarithm of the historical data to obtain coefficients for our exponential functions.

B.1 Costs

B.1.1 Bandwidth Costs

The cost of a transaction on ICTP is a function of data transfer cost. To forecast the cost of bandwidth in the future, we use historical internet transit prices from 1998 for 2015[24] (see Fig. 9).

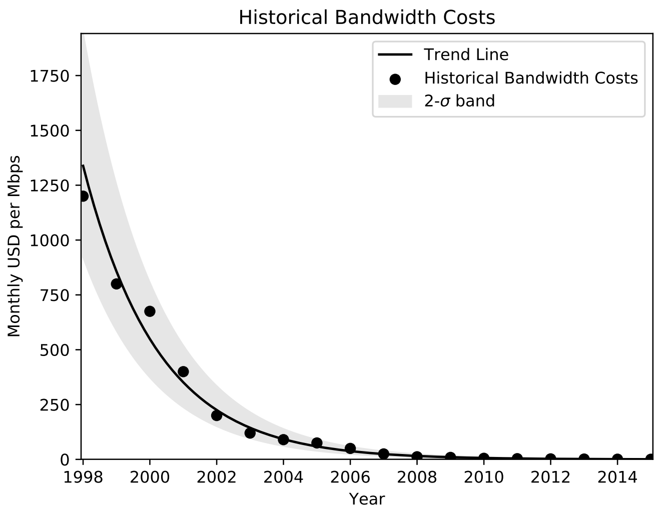

Figure 9: Historical Data Transfer Costs

B.1.2 Storage Costs

The cost of a transaction on ICTP is also affected by storage costs. Again, we rely on historical data to forecast future changes in the cost of storage (see Fig. B.1.2).

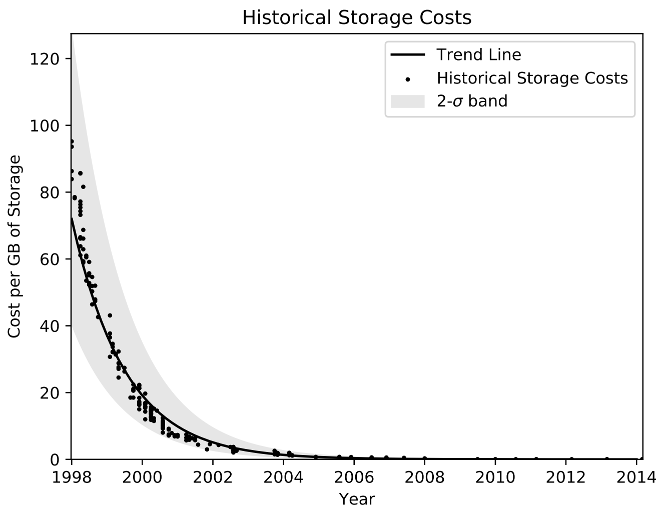

Figure 10: Historical Storage Costs

B.1.3 Total Cost per Transaction

We can project the total cost per transaction using trends in storage and bandwidth costs (see Fig. 11).

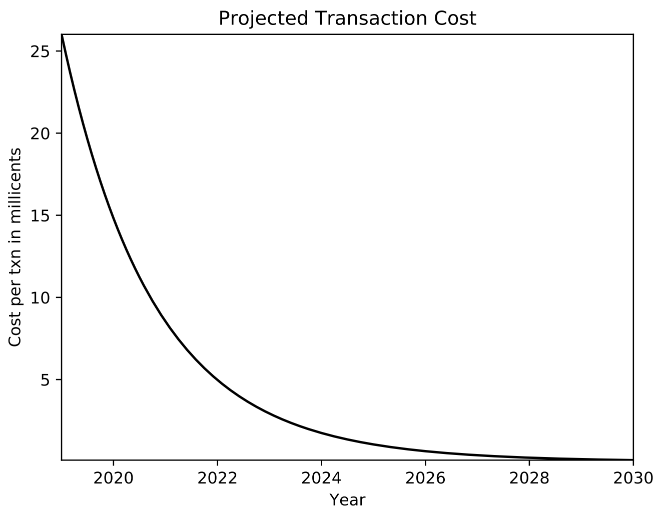

Figure 11: Projected Cost per Transaction

B.2 Capacity

Global capacity is difficult to measure (and also difficult to define), so we will conservatively estimate it with global internet traffic.[19] We rely on historical data (see Fig. 12).[25]

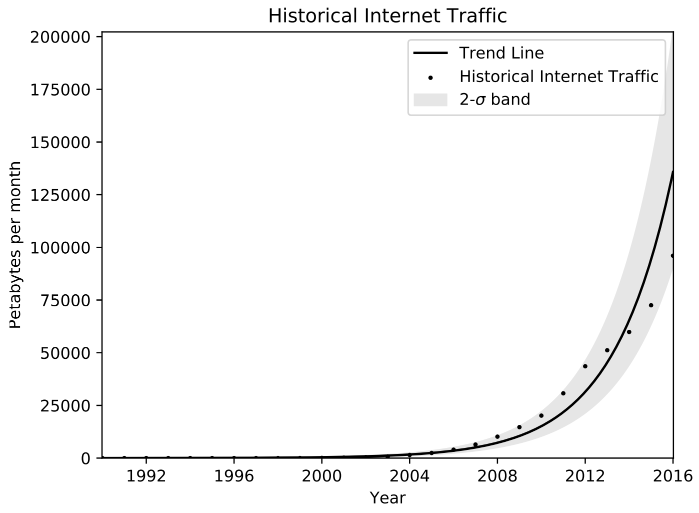

Figure 12: Global Internet Traffic

Figure 13 shows the projected growth in the capacity of the network assuming ICTP will

[19]This does not consider possible bottlenecks such as undersea cables. While it's worth mentioning that utilization of undersea cables tends to be only around 30%, a full analysis is beyond the scope of this paper.

use 42% of global internet traffic.

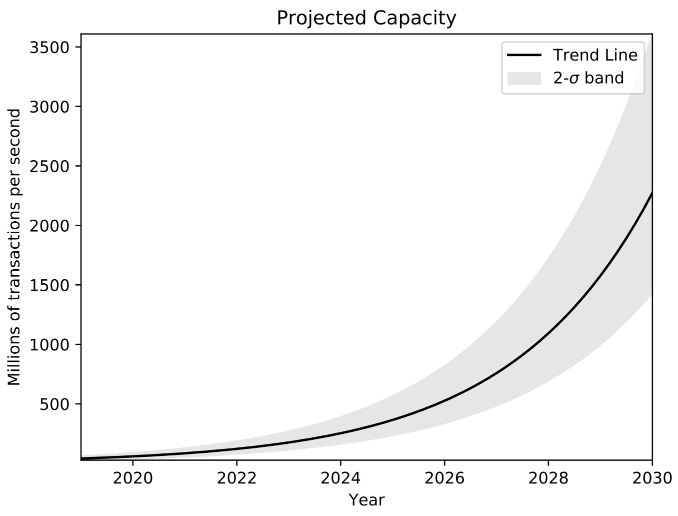

Figure 13: Projected Capacity of ICTP

References

[1] Ralph C Merkle. Method of providing digital signatures, January 5 1982. US Patent 4,309,569.

[2] Donald R Morrison. Patricia—practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM (JACM)*, 15(4):514–534, 1968.

[3] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[4] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 1996.

[5] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[6] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[7] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. Cryptology ePrint Archive, Report 2016/492, 2016. https://eprint.iacr.org/2016/492.

[8] QuantumMechanic. Proof of stake instead of proof of work. https://bitcointalk.org/index.php?topic=27787.0.

[9] Herbert Hellerman. *Digital computer system principles*. McGraw-Hill, 1967.

[10] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[11] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[12] JustinDrake. Leaderless k-of-n random beacon. https://ethresear.ch/t/leaderless-k-of-n-random-beacon/2046.

[13] Peter Robinson. Decentralised random number generation. *arXiv preprint arXiv:1807.09225*, 2018.

[14] Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with webassembly. In *ACM SIGPLAN Notices*, volume 52, pages 185–200. ACM, 2017.

[15] FH Jackson. A generalisation of the functions γ (n) and xn. *Proceedings of the Royal Society of London*, 74:64–72, 1904.

[16] Sandvine. Global internet phenomena Latin America and North America, 2015. https://www.sandvine.com/hubfs/downloads/archive/2015-global-internet-phenomena-report-latin-america-and-north-america.pdf.

[17] Paul Langacker. Grand unified theories and proton decay. *Physics reports*, 72(4):185–385, 1981.

[18] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International Palo Alto, 1979.

[19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018.

[20] Bert Hölldobler and Edward O Wilson. *The superorganism: the beauty, elegance, and strangeness of insect societies*. WW Norton & Company, 2009.

[21] Ian McDougall, Francis H Brown, and John G Fleagle. Stratigraphic placement and age of modern humans from Kibish, Ethiopia. *Nature*, 433(7027):733, 2005.

[22] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru, 2018.

[23] Fabio Caraffini, Ferrante Neri, Benjamin N Passow, and Giovanni Iacca. Re-sampled inheritance search: high performance despite the simplicity. *Soft Computing*, 17(12):2235–2256, 2013.

[24] William B Norton. Internet transit prices-historical and projected. *Report, DRPeering International, Aug*, 2010.

[25] VNI Global. Mobile internet traffic forecasts. https://www.cisco.com/c/en/us/solutions/service-provider/visualnetworking-index-vni/index.html, 2016.