Teller

Automate Trust

Jeevan J. Singh Jaswinder Singh Gabor Levai

Preamble

Distributed ledger technology (DLT) is fundamentally changing the roles of trusted intermediaries in validating transactions. It is replacing large institutions with everyday people and their laptops. This radical shift has the potential to make the entire transactional environment cheaper, faster and much more trustworthy.

DLT's that are operating today are fundamentally deficient; they are slow and costly, making them unusable for real world applications.

Teller is proposing a brand new infrastructure that is scalable, open source and above all, usable.

Overview

Distributed Ledger Technologies (DLT) have the potential to make a massive impact on current payment systems. In their default configuration, DLTs have four fundamental flaws that hinder mass adoption: speed, bandwidth, privacy and governance.

SPEED
Current operating DLT systems like Bitcoin and Ethereum, among many others, have unreasonable settlement times. On-boarding consumers is an impossible task for platforms that are much slower than existing centralised ledgers.

BANDWIDTH
Visa and MasterCard are both capable of processing thousands of transactions per second. In comparison, Bitcoin and Ethereum can process 6 and 15 transactions per second respectively. This lack of throughput creates peak volume constraints that further slows down systems, resulting in transactions that take hours or even days.

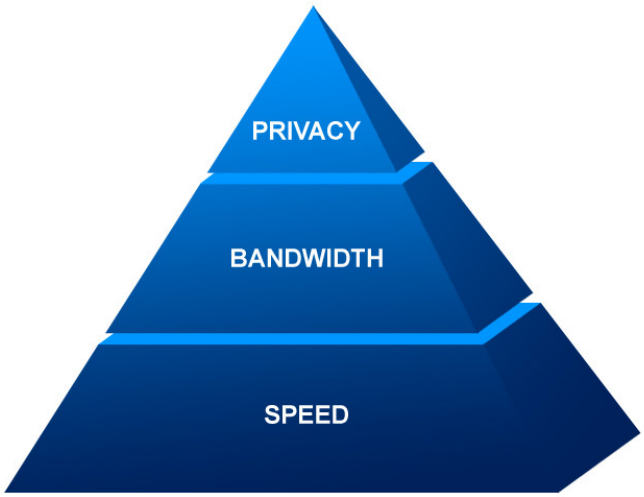
PRIVACY
Typical public DLTs maintain a shared ledger that is fully accessible, allowing anyone to view everyone's day to day transactions. The vast majority of consumers don't want to open up their spending habits to the general public.

Private and permissioned blockchains manage to solve the three issues mentioned above but are hindered by a different issue:

GOVERNANCE
Many companies have swapped their distributed governance model for something more centralized, resulting in massive performance gains. This paradigm shift goes against the very fabric of the distribution revolution that we are currently seeing. In an attempt at centrism, companies are turning to private blockchains, where a small handful of actors govern the system. While this may be satisfactory to some, we believe that distributed ledgers should be governed by the people that invest in and operate it.

If DLTs are to have any measure of success then we must outcompete today's legacy systems in terms of performance. Consumers expect to complete a transaction in mere seconds and DLTs must break through that benchmark.

From a consumer standpoint, what are the keys to success for the mass adoption of DLT's?



In terms of speed, we need to pass the Coffee Test. If a consumer was to buy a coffee with our native currency, how long would they be willing to wait for the transaction to close? It would not be more than a few seconds. Currently public blockchains take far too long.

In terms of bandwidth, if thousands of people bought coffee at the same time, how will the system handle peak demand? Traditional systems like Visa and MasterCard can process 2,000 transactions per second. Bitcoin and Ethereum can process 6 and 15 transactions per second respectively. DLTs need to reach a bandwidth of at least a few thousand transactions per second.

And in terms of privacy, do people really want the world to know their coffee purchasing habits? The pseudo-anonymity provided by current blockchains is insufficient. Details of a transaction should only be known by those directly involved.

Introduction

Teller is a new public peer-to-peer transaction system that solves the fundamental problems that plague DLTs. Teller can close a transaction in milliseconds, has a current bandwidth of 100,000 transactions per second (TPS), and can keep financial data private. Although our expected current bandwidth is substantial enough to run a modern ledger system, we are researching ways to scale it even further. The current limitation of 100,000 is due to time sync constraints between nodes. We are actively exploring solutions that will push our TPS to at least a few million. Teller has an open source governance model that leaves the community in charge.

How does it work?

Traditional blockchains use Proof of Work or Proof of Stake to settle a transaction and the data is appended on a blockchain before it is settled. The longer the chain the longer it takes to settle a transaction.

In any given transaction on the Teller network, there are six actors. The sender, receiver and four randomly selected validators. When a sender starts a transaction, his ledger is validated by the four validators and the transaction is then validated. The data from that transaction is stored on only six nodes instead of the entire network.

Key Innovations

CHAOS GENERATOR

Teller taps into chaos theory to maintain the centralised integrity of the system. We leverage chaotic occurrences in the world, like the movements of jellyfish, to select our four random validators. By unpredictably selecting a few random validators, we reduce the bulkiness of our system without sacrificing trustlessless.

SNAPSHOT VALIDATION

To further improve the speed of our system, we implement a snapshot validation feature. Instead of validating every transaction in a ledger, a hash of the entire ledger is generated and compared against the most recent snapshot of that ledger.

Snapshots are generated when a transaction is considered complete and are then distributed to the parties involved, as well as a set of selected snapshot nodes. These snapshot nodes are selected at regular intervals and assigned to various nodes to perform snapshot validation if requested. When snapshot nodes are reassigned at the regular interval, the snapshots of the nodes that they are assigned to are provided by the previously assigned snapshot nodes.

Through this process, we greatly reduce the computing time required to validate a ledger, and add an extra layer of data redundancy to our network.

FRACTIONAL LEDGER

When a transaction is complete, the entire transactional data is stored with the Sender and Receiver, while only specific parts are stored on the validating nodes. Through the use of Snapshot Validation, and storing only the information required to satisfy validation, we achieve both privacy and trustlessness in our network.

GEMINI FENCING

Teller has two classifications for nodes: regular nodes and partner nodes. Regular nodes are volunteer nodes that do not receive dividends for participating in transaction validation and have limited responsibilities. They can also opt-in to the KYC process. Partner nodes are full nodes that require a full KYC process and vetting from other partner nodes to be admitted in the network. Partner nodes receive a percentage of the transaction fee for transactions that they participate in, either as a validating node or a snapshot node.

Chaos Generator

The core of Teller's chaos number generator relies on chaos theory. Chaotic behaviour occurs in natural systems like weather, climate, and the movements of jellyfish among many others. The wildly random jellyfish movements are just a start as Teller will be integrating many more chaotic elements into its chaos generator. We take pictures of live jellyfish feeds from aquariums around the world and translate them into data sets that in turn generate a random key. The random key is used to determine which nodes will participate in the validation process.

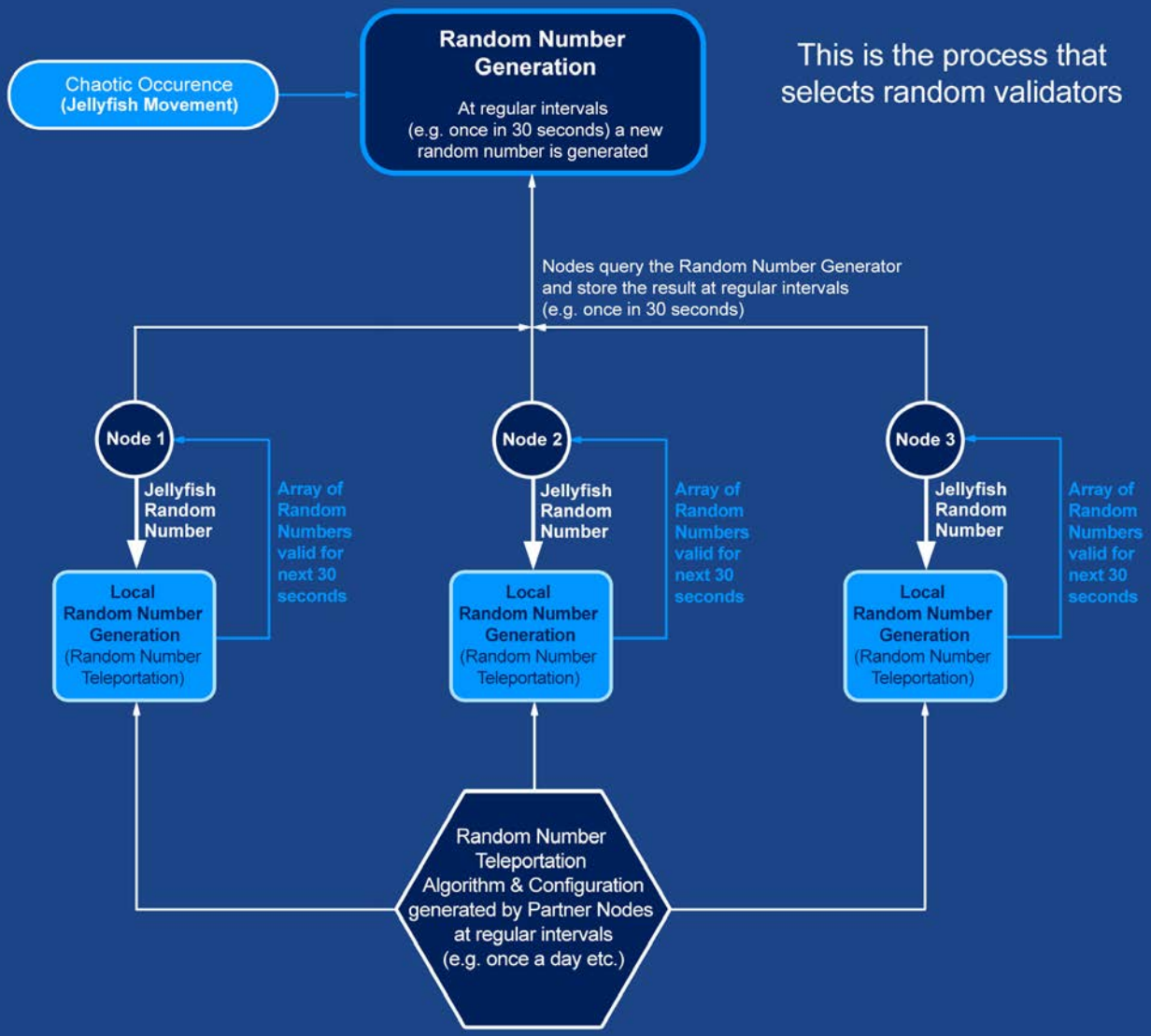
THIS SERVES THREE MAIN PURPOSES:

1. It maintains the decentralised integrity of the system as no one entity can dictate who is part of the process.

2. Since it is impossible to predict movements of a jellyfish, it becomes impossible to determine which nodes will participate in a given transaction. This makes it impossible for anyone to hack into a transaction while it's occurring in an attempt to falsify the datasets.

3. Since validators are different for each transaction, it becomes impossible for someone to falsify their ledger through collusion.

Chaos Generator



Privacy

Teller guarantees privacy. We rely on a very secure one-way encryption algorithm **(SHA-256)**. By storing only the transactional data necessary for validation on nodes involved in a transaction, we can provide a strong level of privacy.

As an example, if Alice sends $100 to Bob, that transaction will involve twelve partner nodes and four regular nodes.

Of the twelve partner nodes:

- 2 will act as Validating Nodes
- 2 will be referenced to validate their role in the most recent transaction in the Sender's ledger
- 8 will act as Snapshot Nodes

Of the four regular nodes:

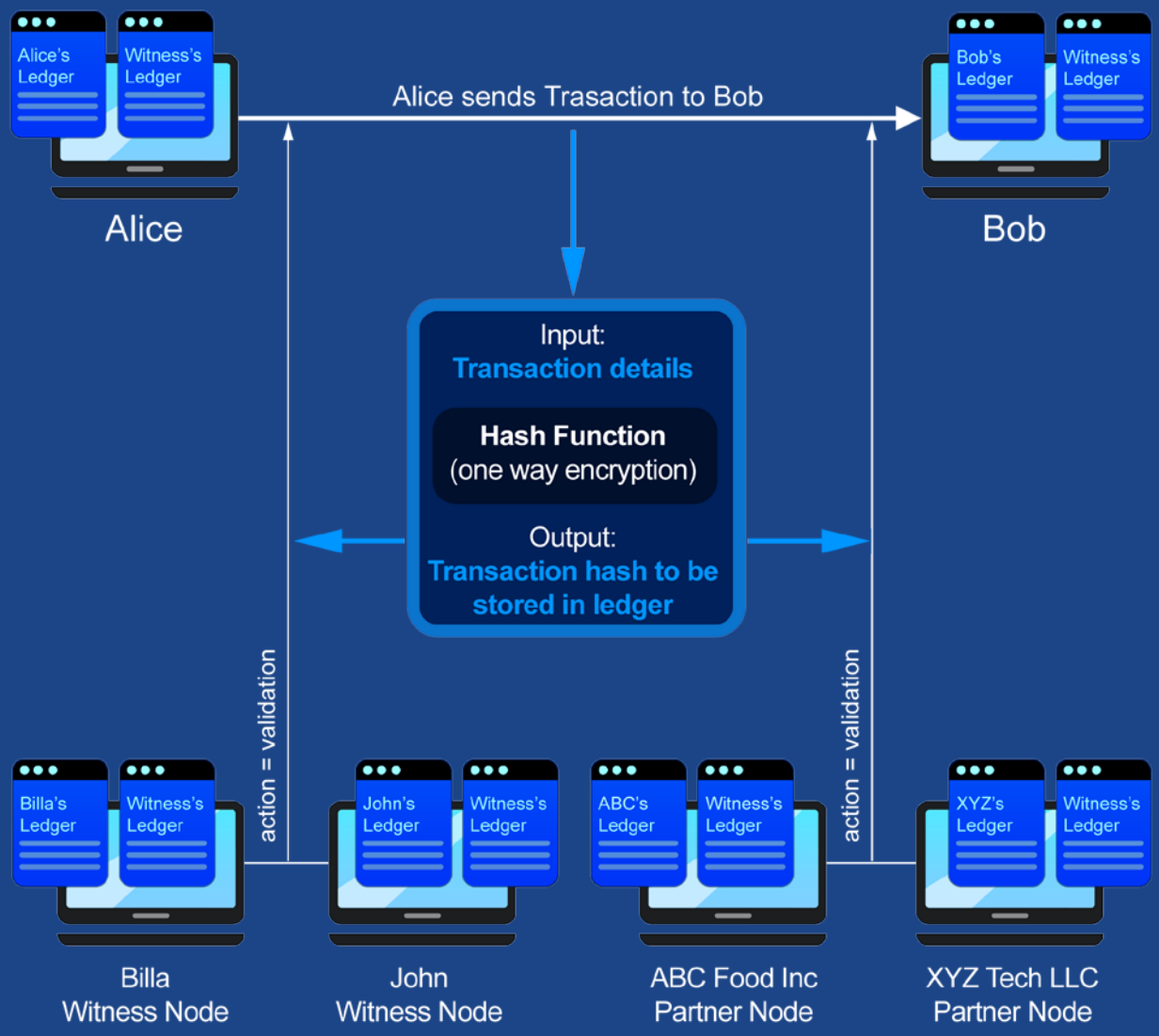
- 2 will act as Witness Nodes
- 2 will be referenced to validate their role in the most recent transaction in the Sender's ledger

If all of these nodes perform their appropriate validation process and come to agreement that the transaction is valid, the following occurs:

- The new valid transaction is appended to both the Sender and Receiver ledger
- A new snapshot is generated
- The new snapshot is distributed to the Snapshot Nodes
- Unnecessary transactional data for future validation is removed
- The transactional data needed for future validation is stored on the Validating Node's witness ledgers

This mechanism maintains the security of traditional blockchain systems while addressing the privacy concerns that come with traditional public blockchains.

Privacy

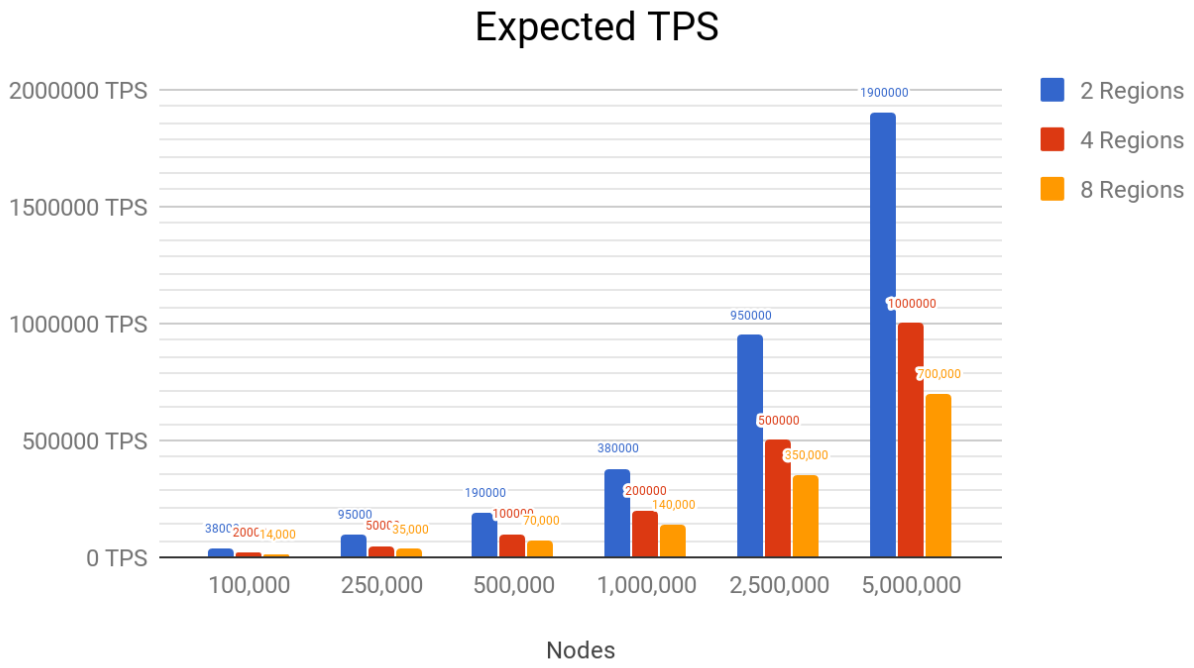

Alice's Ledger
Witness's Ledger
Alice
Alice sends Trasaction to Bob
Bob's Ledger
Witness's Ledger
Bob
Input:
Transaction details
Hash Function
(one way encryption)
Output:
Transaction hash to be stored in ledger
action = validation
action = validation
Billa's Ledger
Witness's Ledger
Billa
Witness Node
John's Ledger
Witness's Ledger
John
Witness Node
ABC's Ledger
Witness's Ledger
ABC Food Inc
Partner Node
XYZ's Ledger
Witness's Ledger
XYZ Tech LLC
Partner Node

Performance – Bandwidth

The following table displays our transactions per second (TPS) results from our testnet with limited node configurations.

Nodes	TPS	TPS	TPS
50	11	10	6
75	21	11	10
100	30	20	13
Regions	**2 Regions**	**4 Regions**	**8 Regions**
Escalation Factor	0.38	0.2	0.14

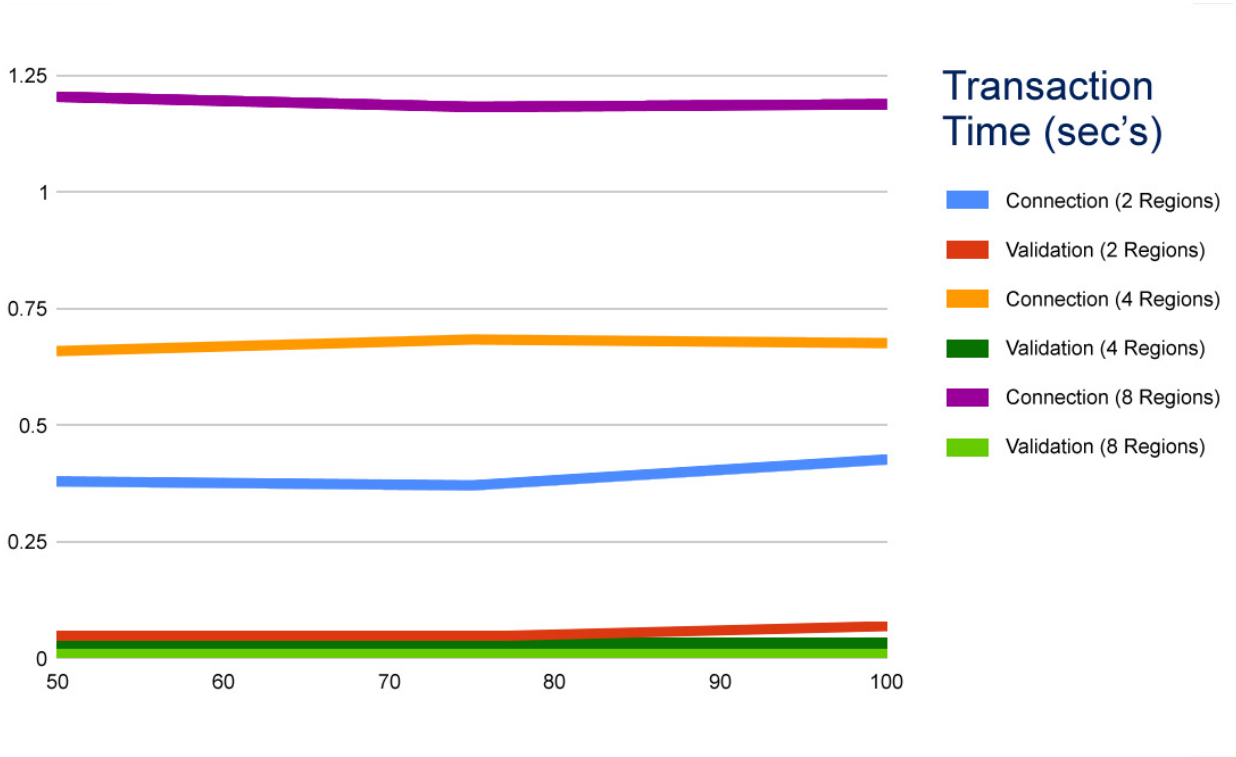
The results have been further extrapolated to project TPS with a higher node configuration. It is to be noted that our current testnet has not been optimized for efficiency; therefore final TPS should be much higher than current results show. The long term goal is to scale the network to 2.5 Million TPS regardless of region. There are a few solutions in the pipeline to that effect.



Performance – Speed

The following are the speed results for Teller testnet. The validation times (validation) are linear regardless of the number of regions. The connection time varies between regions as expected.

Transaction time on the Teller network will vary depending on where the selected validators are located. Minimum time to conclude a transaction is approximately .04 seconds. Maximum time required to conclude a transaction is approximately 1.25 seconds.



Methodology

Our tests were run on a network of Teller nodes hosted on AWS EC2 instances. We ran a series of tests in a variety of configurations to accurately represent the potential of our network.

The Teller network is comprised of **partner** and **regular** nodes. To mimic the suggested minimum requirements for operating a node listed earlier in the white paper, **partner** nodes were hosted on **t2.large** instances, and **regular** nodes were hosted on **t2.medium** instances.

Tests were executed on a selection of 8 regions: Virginia, Oregon, Canada (Central), São Paulo, Sydney, Seoul, Tokyo, and Frankfurt.

The regions selected for the tests listed in the performance section are as follows:

2 Regions: Virginia, Oregon
4 Regions: Virginia, Oregon, São Paulo, Canada (Central)
8 Regions: Virginia, Oregon, São Paulo, Canada (Central), Seoul, Tokyo, Frankfurt, Sydney

Nodes were spread evenly among these regions for accurate testing.

To properly represent our estimated node configuration on a live network, we set the ratio of regular:partner nodes to 5:1.

Using this node configuration, in a network of 10,000 nodes, we would see this type breakdown:

NODES: 10,000
PARTNER: 2,000
REGULAR: 8,000

Of the 8,000 regular nodes in the above example configuration, a quarter of them would be provisioned as sender nodes for the test. This means they **would not** be eligible for validating transactions.

Cost

The Teller network is inexpensive to operate. Teller nodes can be run use existing hardware. The variable cost to validate a transactions ranges from $0.00001744 to $ 0.00004690.

Teller network can validate between 200-500 transactions for a penny.

Security

GEMINI FENCING

Teller fences nodes into two spheres; regular and partner nodes. Teller's network relies exclusively on the partner nodes to maintain our system integrity. Fencing nodes allows the network to treat them both differently in terms of KYC.

Partner nodes will go through a heavy KYC process to deter malicious actors from being part of the network. Teller's ability to pinpoint malicious actors down to their real identity will severely deter anyone from trying to break the system.

CRYPTOGRAPHY

The sensitive information is encrypted as per RSA and SHA-3 hashing standards. Information is properly encrypted before sender attempts to transmit the information. Validator receives and performs the validation and transaction commit is executed. All nodes are required to store a set of public and private keys. These keys are necessary to communicate and keep information integrity. All protocols are developed according to NIST standards and guidelines.

DATA REDUNDANCY

Each partner is required to provide redundancy services for other nodes in the network. When a new node is certified and accepted inside partner network, two partner nodes are selected (through our chaos generator) for which the new node will be responsible to store redundant data. This redundancy operation ensure that there are always a minimum two nodes which will have same copy of data in case there is scheduled or non-scheduled downtime for that partner node.

In the event a node drops off permanently, the redundant node will be used to transpose the inactive node's data onto a new node. The system is designed to have multiple layers of data protection. We are currently exploring multiple different options to enhance the quality of the data redundancy environment.

Security

ASYNCHRONOUS BYZANTINE FAULT TOLERANT (aBFT)

Teller guarantees that at any moment of time all nodes agree on a consensus and it is always the same consensus. This makes Teller have a higher level of tolerance to Byzantine faults. These faults can range from failure of not being present, failure to respond to a request, failure of not receiving a request or a failure due to sending inconsistent/incorrect/misleading data.

Teller's higher tolerance stems from its asynchronous operations. Asynchronous operations are, by definition, meant to be circulating in validating nodes. The information is stored in silos i.e. only stored between sender, receiver and witness nodes. When nodes act as senders, Teller's validation function ensures that the previous transaction hashes were correct in order to ascertain sender's ledger integrity. If the random validating nodes are not able to match the hash function the transaction cannot be completed. This is the method used by our system to detect fraudulent ledgers and malicious nodes. As the number of nodes increase in the Teller network, the odds of successfully committing fraud decrease exponentially.

ACID COMPLIANT

Teller transaction system is fully ACID compliant.

Atomicity: We use the "All or none" protocol to store database information. Final information is stored is stored all once on multiple different nodes.

Consistency: Validation rules are same across all transactions regardless of origin; partner node or regular node. The transaction process (validation + execution) is consistent across the entire platform.

Isolation: All transactions happen in parallel and are isolated from each other.

Durability : The network relies on a combination of trusted and trustless nodes. Durability is promised through redundancy and a guaranteed uptime of 99.99% for partner nodes.

Security

SERVICE DISRUPTION RESILIENCY (DDOS)

Teller's goal is to provide and maintain an exceptional level of service in the face of various faults and challenges. Our network of partner nodes have various methods of defending, detecting and remedying various threats in the network. These include proactively pinpointing bad nodes (hacker and non-compliance) through automatic monitoring and scanning, and communicating with each other in the event of a DDOS-type attack or event that causes a disruption in service.

During the early stages, AI scripts will need some time to learn and adjust automated benchmarks and standards. However, this machine learning is integral in our network and will provide significant benefits. An Intrusion Detection System (IDS) and open source code ensure that these threats can be detected in time. Initially, this may mean that the system creates several false alarms but eventually it will lead to industry-leading security standards.

Know your client (KYC)

Our KYC process will be segmented according to node type; regular or partner node.

Regular nodes will have access to an opt-in KYC process.

Partner nodes will go through a stringent and mandatory two-tier KYC process. Upon initiating a request to join the network, Teller will randomly select five partner nodes that will screen the applicant. A minimum of three of the five screening nodes must agree to let the applicant join before the application is approved for voting by partner nodes at large. After a successful vote, the applicant will be initiated as a new partner node.

Teller will be launched with a pre-configured KYC process. Partner nodes can vote on it anytime to adjust the initial configuration.

Blockchain+ will develop the initial tools to automate the entire KYC process. This will also be open source and subject to change as per membership vote.

Blockchain+ will manage the onboarding process for the initial nodes but will remove itself from the process once the Teller network is able to be self sustaining. The following is a schedule of by which Blockchain+ operated nodes will onboard partner nodes vs onboarded partner nodes.

Total Partner Nodes	Blockchain+ Nodes	Partner Nodes
0 - 500	5	0
501 - 2,500	4	1
2,501 - 5,000	3	2
5,001 - 7,500	2	3
7,501 - 10,000	1	4
10,001+	0	5

Governance

Teller is an open source project and will be governed by its stakeholders in both currency and operations. Teller believes in the value brought forth by both the investors and operators of the Teller ecosystem. A node operator without the means to buy a massive amount of stake should not be excluded from deciding the direction of the project. Conversely, a stakeholder should not be forced to proxy their vote because they don't operate a partner node.

Blockchain+ is the for-profit parent company of Teller. Blockchain+ will ensure the growth of the ecosystem through sales, investment in the ecosystem and the acquisition of patents. Blockchain+ will never enforce its patents but will use them as collective protection against patent trolls.

Economics

Teller will operate using a native currency called Tell. Users will be charged a transaction fee to perform transactions on the platform. The validating partners will need to be paid for their services and Blockchain+ will need to generate cash flow to ensure the optimal working of Teller's ecosystem.

The fee will be split in a 75-25 ratio: 75% will go to the validating partners and 25% will go to Blockchain+. The transactional proceeds will be deposited into an escrow wallet for a period of 31 days to deal with any transactions reversals i.e. Scott buys a T-shirt and decides to return it a few days later. After the escrow period, the funds will be transferred to all parties at the same time.

Conclusion

Teller solves the three fundamental problems (speed, bandwidth, privacy) of traditional blockchain systems without compromising open source governance. Teller's fully transparent and community driven model is the foundation of future peer-to-peer transactional systems.

Appendix 1

Technical Paper

Teller

Automate Trust
https://blockchainplus.co

By Gabor Levai, Jaswinder Singh, Jeevan J. Singh

Index Terms—Blockchain, Cryptocurrency, Bitcoin, Ethereum, Hyperledger

Abstract

The following research paper is exploring a new distributed ledger system to build the next-generation peer-to-peer transaction system and solve challenges faced by current blockchain solutions. The cryptocurrency (crypto) market has seen a massive spike in value due to the proliferation of user and investor interest. The nuclear rise of crypto users has proven that decentralized currencies will be part of the future payment ecosystem. However, the technology platforms on which these currencies operate are fundamentally deficient. The need for public blockchains to store data on millions of nodes creates three main problems:

1) Users are not comfortable making their data public on every single node regardless of the robustness of the encryption mechanism,
2) Both the transaction time and the resources it takes to mine blocks and create new blocks does not always fit the turnaround time promised to their users,
3) This resource intensive process also limits the bandwidth of the network (transactions per second) to double digit numbers.

Private blockchains are an alternative that alleviate much of the concerns inherent to public blockchains. However, despite being a blockchain solution, they still inherit the problems of centralized permissioned systems, i.e. lack of trustless-ness and a single point of failure. In this paper we will be exploring the use case of a consortium blockchain to solve the three main problems of traditional public blockchain networks as outlined above.

Teller Open Source Community

The architecture for any peer-to-peer transaction system needs to be open source. The trustless aspect of any blockchain system emerges from a lack of a central authority dictating any operational aspect. However, initially there needs to be a central party to help manage initial day to day operational aspects. Teller's parent company Blockchain+ will manage operations during Teller's infancy. Blockchain+ will hand over all operational aspect to the community once Teller matures and is self-sustaining.

Architecture

Nodes

Teller defines nodes as a computational entity or a point of communication that can perform transactions. Nodes are distributed databases which transmit information within the Teller network to perform transactions or housekeeping tasks. There are two types of nodes: regular and partner:

Regular nodes

Regular nodes are lightweight nodes, such as web browsers and applications (Mobile or Desktop). Regular nodes perform ledger operations, such as sending and receiving. They are controlled by end users; hence, their uptime is not guaranteed. Regular nodes are not involved in activities to maintain network integrity.

Partner nodes

These nodes have strict minimum hardware specifications: 8GB RAM, 100GB disk,4.0 MBPs, and a guaranteed uptime of >99.00%. Partner nodes must be approved and certified by its peers before it is assigned a place in the Partner subnetwork. Partner Nodes will also have 80 yearly hours of scheduled and unscheduled downtime to cover any hardware and software failings.

Partner nodes are involved in housekeeping tasks, such as updating and distributing random number "teleportation", algorithm generation, and a set of configuration data. Partner nodes actively look for threats and scan for bad Partner nodes i.e. Partner node A falls below the 99% uptime threshold and will be reported to the network by Partner node B. After the report and with the consensus of the network the certification of the partner can be revoked by Teller's Partner node community. These housekeeping tasks are important to maintain the quality and integrity of Teller's network.

Sender and Receiver Nodes

Any node can act as a sender or receiver node in the network. Partner nodes are treated as regular nodes when they send or receive a transaction.

Witness Nodes

The term "witness node" define nodes which are witnessing the current transaction i.e. validating nodes. In addition to that, it also defines previous validating nodes of sender's last transaction. Hence term validating nodes is a subset of witness node however to avoid confusion, terms validating node and witness node are used interchangeably when original validating nodes are being discussed.

Witness Node Pair (Regular)

Two randomly selected regular nodes participate in every transaction. They are responsible for validating the sender's my_ledger transactions and performing the transaction validation along with the partner nodes.

Witness Node Pair (Partner)

Two randomly selected Partner nodes participate in every transaction. Guaranteed uptime is crucial in validating previous transactions therefore the need for a partner node pair.

Data Structure

Every node database consists multiple data sets. Its own ledger, witnessed ledgers (witness_ledger), configuration files, and random number algorithm table (algo_table). Below is detailed representation of these data sets.

My_ledger **Witness_ledger**

Fields	**Fields**
Transaction ID	Transaction ID
From	From
Amount	To
To	Witness1
Witness1	Partner1
Partner1	Witness2
Witness2	Partner2
Partner2	Signature
Signature	Timestamp
Timestamp	

Note that witness_ledger does not store "Amount". Witness and Partner Nodes will only know the amount during the transaction. Afterwards, the amount information will be available to the sender and receiver only. Therefore, any subsequent node sending an inquiry to Witness and Partner nodes for transaction validation will not know the amount. The validation will be done by checking the signature. This important feature ensures transactional privacy. Below is other data sets stored in the node's database.

Data Set Name	**Fields**		
my_data	**key**	**Value**	
	myID	INTEGER	
	login_id	INTEGER	
	key_x	value_x	
random_node	**Random number generated**	**Timestamp**	**AlgoId**
	Random_Number_1	Timestamp_1	AlgoId_1
	Random_Number_2	Timestamp_2	AlgoId_2
	Random_Number_x	Timestamp_x	AlgoId_x
algo	AlgoId	Algo_value	
config	**key**	**Value**	
	key_x	value_x	

Depending on space requirements, Node dataset will store x months of data. Algo_table will get updated on a regular basis, such as once a day.

Partner_data	**key**	**Value**
	Partner_certificate	String
	Last_login_id_in_regular_nodes_network	Integer
Partner_snapshot_data	**key**	**Value**
	NodeId	Integer
	Snapshot Hash	String
	Timestamp	DateTime

Last_login_id_in_regular_nodes_network is the NodeID of the last node that has joined the network. This id is used by the algo_generation function to define a pool to find out which regular nodes are "fresh nodes" meaning they have higher chances stay active during transaction. The idea is that only nodes that just logged in (fresh node) are used when randomly selecting a witness during the transaction. It assumes that

some old nodes might become deactivated or idle for some time, which will result in choosing witness nodes.

Ledger Blockchain

For each node, regardless of type, ledgers are connected to one another through a blockchain. PrevHash is used to make connections and Signature is used to validate the integrity of that block. PrevHash is a hash value of the previous block, and signatures are created with the node's private key of the current block. Key considerations when creating our blockchain are:

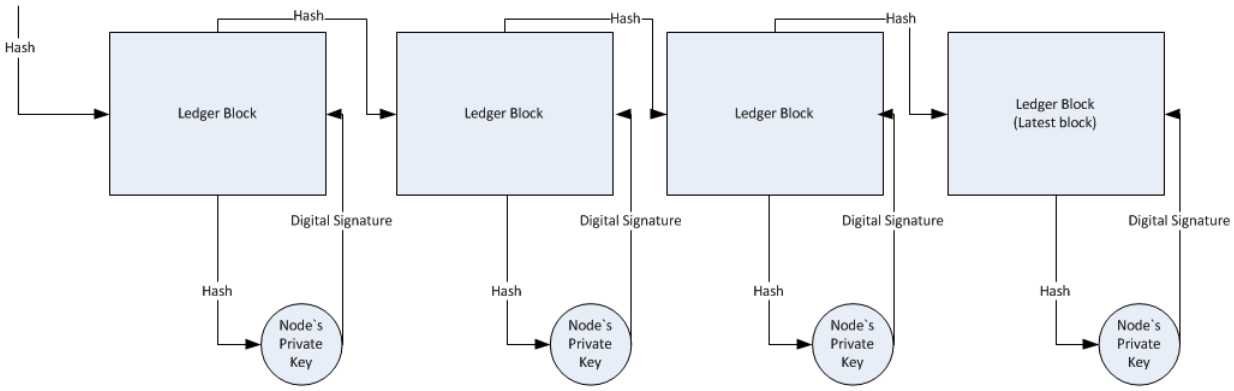
(a) The sender and receiver will have different signatures because of their different private keys;

(b) The purpose of the signature is twofold:

(i) Any node can use the signature to validate its own blocks in the ledger and check if there are any hacking attempts. This assumes that the private key will be securely stored outside the ledger. Securing a private key is not entirely in the scope of this research paper; however, there are several methods available;

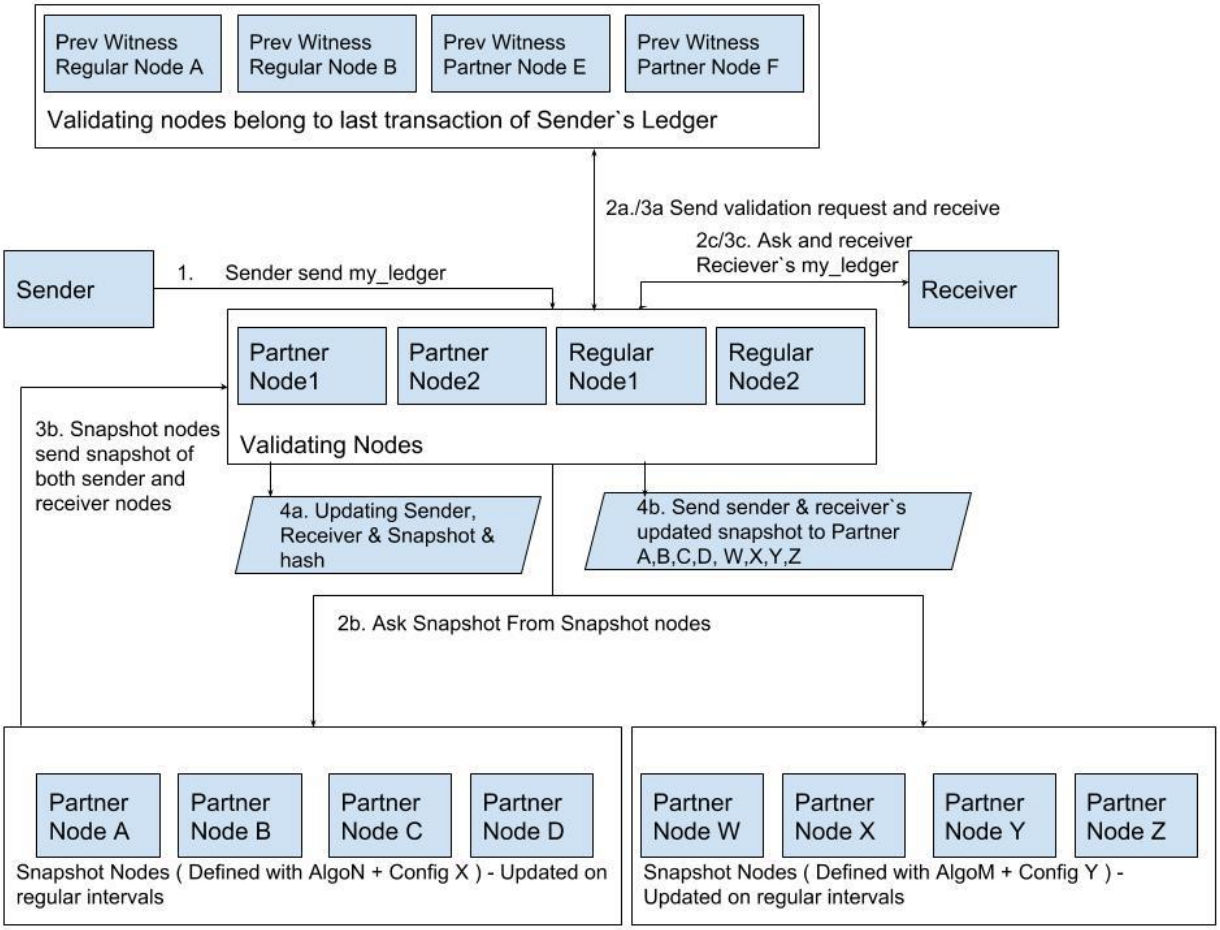
(ii) If a dishonest node alters its own ledger and recalculates hashes and signatures, its next transactions will be invalidated during the execution process of establishing a valid proof (transaction validation) because the signatures or hashes are stored in the witness' witness_ledger as well;

(c) The hash of the current block does not need to be stored, as it can be easily calculated with the hash function.



Transaction Flow

Executing a transaction in the Teller network is a multi-step process. It starts with initiation, then witness/Partner pair selection, validation, transaction id calculation, and finally, execution. The following diagram gives a high-level overview of a transaction workflow.



The following are the steps and an in-depth breakdown of a transaction workflow.

Workflow Steps

Transaction Initiation

A transaction is initiated when the sender node N_S sends one Teller Coin C_x to the receiver node N_R. Before node N_R commits the transaction in its ledger, it is important that validation of the sender must happen first. Sender starts the next step which will generate a randomly selected witness/partner pair.

Witness Partner Pair Selection

A random number is used to decide which Regular and Partner nodes will act as witnesses to perform the validation. The function “random number teleportation” is being called, which returns a set of random numbers for the next period in which they are valid. These random numbers are then consumed by the following functions.

F_xConvertRandomNumberToNodeNumber ($Node_N$) matches the node number to the random number. This function is called four times to select two pairs of Regular Nodes and two pairs of Partner Nodes, as shown below.

Set of Regular Nodes

$NodeRegular_1$ = F_xConvertRandomNumberToNodeNumber (false)
$NodeRegular_2$ = F_xConvertRandomNumberToNodeNumber (false)

Set of Partner Nodes

$NodePartner_1$ = F_xConvertRandomNumberToNodeNumber (true)
$NodePartner_2$ = F_xConvertRandomNumberToNodeNumber (true)

Due to the high volatility built into Teller's random number generator, it becomes mathematically impossible for known nodes to validate transactions for each other. This eliminates the risk of fraud through collusion.

Since regular nodes are volunteers, Teller must account for some chosen nodes not being online. If that happens, another random number will be generated to select new nodes, and this will repeat itself until all four nodes are online. The wait time to generate new nodes is currently configured at 1 second but can be adjusted. Partner nodes have a guaranteed uptime of 99%, therefore finding them offline during a transaction will be an extremely rare occurrence. However, if it happens, the regular node process will be applied to them as well.

Additionally, a regular node may go offline during a transaction, as it is completely volunteering its resources. There are a few ways to mitigate this risk. For example, the user belonging to that regular node can be notified or the validation process may fail for that node; however, as discussed in next topic (Validation), in some cases, it might not pose a problem, as it is expected that the regular node's state cannot be guaranteed; hence, they are not Partner nodes, but merely regular nodes.

This process selects Regular and Partner nodes for the next x seconds (configurable in Random number teleportation function); however, some unit of time will be "lost" during computation and the selection of nodes. The validation step may need to look back to the regular and Partner selection function to get a fresh array of updated nodes if nodes are unable to fulfill validation criteria.

Validation Preparation

Until now, we have identified six parties which will take part in the transaction.

$Node_S$ = Sender
$Node_R$ = Receiver
$Node_{W1}$ = Regular_Partner_Nodes [0];
$Node_{W2}$ = Regular_Partner_Nodes [1];
$Node_{G1}$ = Regular_Partner_Nodes [2];
$Node_{G2}$ = Regular_Partner_Nodes [3];

*Here, $Node_v$ represents validating nodes, which are a combination of $Node_{W1}$ $Node_{W2}$ $Node_{G1}$ and $Node_{G2}$
For this transaction to be executed, the Regular and Partner node set must validate the ledger of $Node_S$. This step is crucial to make sure that the sender does not send more coins than it has.

Validating $Node_S$ Ledger

In the following steps, we are using (2) (2a) etc. to denote asynchronous transactions taking place in parallel.
1) Sender sends a copy of my_ledger to validating nodes. The validating nodes only need its last transaction detail (minus amount and data field) + snapshot hash. However, it is important that the sender sends my_ledger because if validating nodes generate hash from the sender's my_ledger then it is more reliable method than the sender generating its own hash and transmitting as the sender can "fake" the hash. Although even after "faking" the hash the chances of committing fraud are extremely low because of further checks and balances. The receiver also needs to send my_ledger so that validating nodes can calculate snapshot hash and transmit to snapshot nodes and the not receiver itself.
2.a. Validating nodes start validating the last transaction with previous witness nodes so there will be four checks. Each validating node checking one randomly decided previous witness node.
2. c. This step is asynchronous to step 2a. Validating nodes ask receiver for it's my_ledger
3. a. Results from witness nodes are transmitted back to validating nodes.
3.b. This step is asynchronous to step 3a. Results from snapshot nodes transmitted back to validating nodes.
3.c. This step is asynchronous to step 3a. Receiver node sends my_ledger to validating nodes
4.a. After validation rules are executed (calculate final amount to make sure sender has enough funds to transmit) in addition, the sender's and receiver's ledger are updated with transaction hash and latest snapshot.
4.b. This step is asynchronous to step 4a. Sender's and receiver node's snapshot are transmitted to snapshot nodes.

Please note that snapshot nodes are not randomly selected. Each node has four partner nodes selected as snapshot node which contain snapshot of my_ledger of that node. This is done with help of algorithm + configuration data. The algorithm change on periodic basis and is updated by partner network. This algorithm is different from other algorithms we use (e.g. random number teleportation algorithm) however all algorithms will be updated on a regular non-predefined basis as per partner network request. It is important to note that all old algorithm configuration set are stored on all nodes so that we can look back in time and find out which algorithm was in effect at that point of time. This data will not consume significant disk space as the history data will be very lightweight (as its not generated at high frequency). It is estimated to be around or less than 10kb and will be generated monthly or bi-weekly etc. A 10 years data would be around 2.4MB.

Please also note that we are using validation approach which is combining two different approaches (snapshot check + last transaction check). Both should be used in conjunction of each other. The reason is that snapshot check alone, though looks sufficient to validate the sender/receiver but by itself is not secure enough. Teller will also use a last transaction check to make sure the validation process is secure enough to meet Teller's lofty expectation.

Transaction ID calculation

Before initiating the final step of transaction execution, it is important to provide a unique ID for future identification purposes. However, the main issue is computing a unique ID for two ledgers. Essentially, Node_S.my_ledger and Node_R.my_ledger may have different IDs at the bottom of their queue stack, and any of the following possibilities may be true.

$$\text{Node}_S\text{.my\_ledger.Last\_TransactionID} > \text{Node}_R\text{.my\_ledger.Last\_TransactionID}$$
$$\text{Node}_S\text{.my\_ledger.Last\_TransactionID} < \text{Node}_R\text{.my\_ledger.Last\_TransactionID}$$
$$\text{Node}_S\text{.my\_ledger.Last\_TransactionID} = \text{Node}_R\text{.my\_ledger.Last\_TransactionID}$$

Unless they are identical in ledger length and ID, which is a rare possibility, assigning unique IDs may be challenging, given the fact that the TransactionID for a particular transaction needs to be exactly the same and unique. Therefore, to solve this problem, we propose a mutually-signed solution which will ensure the identicalness and uniqueness of the TransactionID stays intact in both ledgers.

Last_Sender_ID = TypeOf INT Subtring (Node_S.my_ledger.Last_TransactionID, 0, "-");
Last_Reciver_ID = TypeOf INT Subtring (Node_R.my_ledger.Last_TransactionID, 0, "-");
NewID = Contact (++Last_Sender_ID , "-" , ++Last_Reciver_ID);

Transaction Execution

The transaction is executed by logging information in my_ledger of both Node_S and Node_R. The hash is computed among Node_{VX} in consensus and logged in both my_ledger accounts. Node_{VX} also updates their witness_ledger with this hash so that it is available for future validation purposes. Additionally, a my_ledger hash is also calculated by taking a snapshot of my_ledger of sender & receiver nodes.

This information is transmitted to set of corresponding partner nodes. Please note that it is possible that if receiver node is inactive or offline during the transaction execution it is not possible to commit the transaction. However, both partner nodes can hold up the final execution of the transaction for x amount of time before performing final commit. This provides flexibility for the receiver node.

However, after the time period the transaction will be dissolved and get nullified. This operation brings a bit of complexity as the other two regular nodes (witness nodes) will be required to be online at the time of final execution. In case they are not online then the two partner nodes will need to issue a "non-sufficient-nodes-available-for-transaction" call back to sender and receiver. This will prompt the sender to generate new set of validating nodes to restart the validation cycle. Since both sender and receiver will be online during the new call, the transaction will be settled without delay.

Teller in Action

In this section, we will discuss real-life working models. The process will start by assuming that the 1st Teller transaction can be done without verification. This will initiate the transaction cycle, and all other nodes will borrow money from the very first node.

Actors

Node Identification	Type of Node	Comment
Alex	Regular Node	Regular nodes operating as light nodes
John	Regular Node	
Gogo	Regular Node	
William	Regular Node	
David	Regular Node	
Robert	Regular Node	
Beera	Partner Node	Partner nodes set up by individuals via VPS, etc.
Michael	Partner Node	
ABC Corporation	Partner Node	Partner nodes set up by small or enterprise firms.
XYZ Corporation	Partner Node	

Transactions

Transactions are color-coded for readability purposes. There are five transactions with different colors assigned to them (white, orange, yellow, green, and blue). Crossed-out nodes indicate their ineligibility to take part in the transaction, which is explained in the comments. A column No is created to explain comments related to each attempted transaction. Please also note that only the relevant columns are shown. Other columns, such as timestamp, are not shown to minimize the data in the example.

No	ID	Sender	Amount	Receiver	Witness1	Partner1	Witness2	Partner2
1	1001_1001	Alex	100	John	Gogo	Beera	Robert	ABC Corporation
2		~~John~~	~~10~~	~~Robert~~	~~Alex~~	~~ABC Corporation~~	~~Gogo~~	~~Beera~~
3	1002_1001	John	10	Robert	~~Alex~~	ABC Corporation	~~Gogo~~	Michael
4	1003_1001	John	10	William	David	~~Beera~~	~~Robert~~	~~Michael~~
5	1003_1001	John	10	William	~~David~~	Michael	~~Robert~~	XYZ Corporation
6	1002_1002	Robert	8	William	~~John~~	Beera	~~Gogo~~	XYZ Corporation
7		John	81	Gogo	William	ABC Corporation	Robert	Michael

No	Result	Comments
1	Processed	First transaction processed without any validation. Snapshot of Alex & John is updated at corresponding partner nodes (P1,P2..PN) and (N1, N2,..Nn) respectively.
2	Insufficient participant during validation	Snapshot of John was validated from partner node (N1,N2..Nn)
3	Processed	Snapshot of John & Robert was validated from partner node (N1,N2..Nn) & (G1,G2...Gn) respectively. Alex and Congo are unable to validate, so we need new validators
4	Insufficient participant during validation	Snapshot of John was validated from partner node (N1,N2..Nn). Only one eligible Partner. We need two.
5	Processed	
6		Snapshot of Robert was validated from partner node (Pr1,Pr2..Prn). Two Partner nodes (Beera, XYZ Corporation) validated and there was only one transaction, so John/Gogo weren't needed
7		Updated snapshot of John,Gogo is committed to partner node (N1,N2..Nn) & (Y1,Y2..Yn)

Validation Process

Following is a detailed validation process of the above transactions. Each row represents an attempted validation step. Some became invalid, and others were then iterated with some variation. The next table (Comments) provides the details of each corresponding validation step.

No	ID	Sender	Amount	Receiver	Witness1	Partner1	Witness2	Partner2
1	1001_1001	Alex	100	John	Gogo	Beera	Robert	ABC Corporation
2		~~John~~	~~10~~	~~Robert~~	~~Alex~~	~~ABC Corporation~~	~~Gogo~~	~~Beera~~
3	1002_1001	John	10	Robert	~~Alex~~	ABC Corporation	~~Gogo~~	Michael
4	1003_1001	John	10	William	David	~~Beera~~	~~Robert~~	~~Michael~~
5	1003_1001	John	10	William	~~David~~	Michael	~~Robert~~	XYZ Corporation
6	1002_1002	Robert	8	William	~~John~~	Beera	~~Gogo~~	XYZ Corporation
7		John	81	Gogo	William	ABC Corporation	Robert	Michael

No	Result	Comments
1	Processed	First transaction processed without any validation. Snapshot of Alex & John is updated at corresponding partner nodes (P1,P2..PN) and (N1, N2,..Nn) respectively.
2	Insufficient participant during validation	Snapshot of John was validated from partner node (N1,N2..Nn)
3	Processed	Snapshot of John & Robert was validated from partner node (N1,N2..Nn) & (G1,G2...Gn) respectively. Alex and Congo are unable to validate, so we need new validators
4	Insufficient participant during validation	Snapshot of John was validated from partner node (N1,N2..Nn). Only one eligible Partner. We need two.
5	Processed	
6		Snapshot of Robert was validated from partner node (Pr1,Pr2..Prn). Two Partner nodes (Beera, XYZ Corporation) validated and there was only one transaction, so John/Gogo weren't needed
7		Updated snapshot of John,Gogo is committed to partner node (N1,N2..Nn) & (Y1,Y2..Yn) respectively.

Tx ID to validate	Status	Done By	Checking Ledger of	Checking with all nodes to make sure hash is correct			
1001_1001	INVALID	~~Alex~~	John	Gogo	Beera	~~Robert~~	ABC Corporation
1001_1001	INVALID	~~Robert~~	John	Gogo	Beera		ABC Corporation
1001_1001	PASS	ABC Corporation	John	Gogo	Beera		
1001_1001	INSUFFICIENT	~~Gogo~~	John		Beera		~~ABC Corporation~~
	Get new validator nodes. They do not need to validate already validated transactions, but we need to replace nodes which are unfit to validate. Alex, Robert, and Gogo must go.						
1001_1001	PASS	Michael	John	Gogo	Beera		ABC Corporation
- Because both the nodes ABC Corporation and Michael are Partner nodes, we are fine as of now. At least 2 Partner nodes give blessing to a single transaction. - We needed 2 checking agents because the ledger has only 1 transaction. If there is more than one transaction, then one agent is sufficient.							
1001_1001	NOT ELIGIBLE	Beera	John	~~Gogo~~	~~Beera~~	~~Robert~~	ABC Corporation
1001_1001	PASS	Michael	John	~~Gogo~~	~~Beera~~	~~Robert~~	ABC Corporation
1002_1001	INSUFFICIENT	Michael	John	~~Alex~~	ABC Corporation	~~Gogo~~	~~Michael~~
Now we need to remove and add another Partner because in every transaction we need two valid Partners.							
Get new validator nodes.							
1002_1001	PASS	XYZ Corporation	John	~~Alex~~	ABC Corporation	~~Gogo~~	Michael
1002_1001	NOT ELIGIBLE	David	John	Alex	ABC Corporation	~~Gogo~~	Michael
1002_1001	PASS	XYZ Corporation	Robert	~~Alex~~	ABC Corporation	~~Gogo~~	Michael
1002_1001	PASS	Beera	Robert	~~Alex~~	ABC Corporation	~~Gogo~~	Michael
1001_1001	PASS	William	John	Gogo	Beera	Robert	ABC Corporation
1002_1001	INSUFFICIENT	ABC Corporation	John	~~Alex~~	ABC Corporation	~~Gogo~~	Michael
1003_1001	PASS	Michael	John	David	Beera	~~Robert~~	~~Michael~~

Validation Steps Explained

Tx ID	Comment
1001_1001	Alex can act as validator, but not for this transaction.
1001_1001	Robert (sender) should never act as a validator.
1001_1001	- Please note that even though ABC corporation was a Partner node in the same transaction it tried to validate, there is no problem because Gogo and Beera have answered it as true, so it can validate. - We need at least 2 checking agents per transaction.
1001_1001	Randomly selected from 4.
	Get new validator nodes. They do not need to validate already validated transactions, but we need to replace nodes which are unfit to validate. Alex, Robert, and Gogo must go.
1001_1001	We need at least 2 checking agents per transaction. - Validation Complete
	- Because both the nodes ABC Corporation and Michael are Partner nodes, we are fine as of now. At least 2 Partner nodes give blessing to a single transaction. - We needed 2 checking agents because the ledger has only 1 transaction. If there is more than 1 transaction, than one agent is sufficient.
1001_1001	Beera cannot validate as it was a participant and it is a Partner node.
1001_1001	Beera cannot validate as it was a participant and it is a Partner node.
1002_1001	Alex/Gogo were ineligible. ABC Corporation replied good. Michael is not eligible because he was a Partner in the transaction where it is validating. Result Insufficient validators. Now this case should not happen in the first place, as Michael already validated the first transaction, but this is only to demonstrate that we need two Partner nodes for validation.
	Now we need to remove and add another Partner because in every transaction we need two valid Partners. Get new validator nodes.
1002_1001	2 Partner nodes passed so result is passed. - validation complete
1002_1001	This should not happen because we need 2 Partner nodes to act as a validator in each transaction, so in this case, even though 2 Partner nodes passed and 1 regular node passed, the result is not eligible.
1002_1001	Alex and Gogo weren't eligible to act as Partner nodes in 1002_1001 so they did not store the transaction.
1002_1001	Two Partner nodes replied YES which completed validation.
1001_1001	Pass
1002_1001	Insufficient players because Alex/Gogo did not take part in validation and ABC Corporation cannot check against ABC corporation. Need at least 2 agents to reply back with pass check. Right now it can only do this against Michael.
1003_1001	Pass - Validation Complete

The following sections describe ledgers of participant nodes. This includes "my_ledger" and "witness_ledger". The same color codes are used for readability purposes.

Ledger : Alex

my_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Signature**
1001_1001	Alex	100	John	Gogo	Beera	Robert	ABC Corporation	12345

witness_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Sign_reciever**

Ledger : John

my_ledger									
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Signature**	**Comment**
1001_1001	Alex	100	John	Gogo	Beera	Robert	ABC Corporation	12345	
	John	~~10~~	Robert	Alex	ABC Corporation	~~Gogo~~	Beera		Not stored in ledger, it failed validation.
1002_1001	John	10	Robert	Alex	ABC Corporation	~~Gogo~~	Michael	81723	
1003_1001	John	10	William	David	Beera	Robert	Gogo	42983	
	John	81	Gogo	William	ABC Corporation	Robert	Michael		

witness_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Sign_reciever**

Ledger : Gogo

my_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Signature**

witness_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Sign_reciever**
1001_1001	Alex	100	John	Gogo	Beera	Robert	ABC Corporation	12345

Ledger : William

my_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Signature**
1003_1001	John	10	William	David	Beera	Robert	Gogo	42983
1002_1002	Robert	8	William	John	Beera	Gogo	ABC Corporation	21389

witness_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Sign_reciever**

Ledger : David

my_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Signature**

witness_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Sign_reciever**

Ledger : Beera - Partner

my_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Signature**

witness_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Sign_reciever**
1001_1001	Alex	100	John	Gogo	12345	Robert	ABC Corporation	12345
1002_1002	Robert	8	William	John	Beera	Gogo	ABC Corporation	21389

Ledger : Michael - Partner

my_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Signature**

witness_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Sign_reciever**
1002_1001	John	10	Robert	~~Alex~~	ABC Corporation	~~Gogo~~	Michael	81723
1003_1001	John	10	William	David	Beera	Robert	Gogo	42983

Ledger: Robert

my_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Signature**
1002_1001	John	10	Robert	Alex	ABC Corporation	Gogo	Michael	81723
1002_1002	Robert	8	William	John	Beera	Gogo	ABC Corporation	21389

witness_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Sign_reciever**
1001_1001	Alex	100	John	Gogo	Beera	Robert	ABC Corporation	12345

Ledger : ABC Corporation - Partner

my_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Signature**

witness_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Sign_reciever**
1001_1001	Alex	100	John	Gogo	Beera	Robert	ABC Corporation	12345
1002_1001	John	10	Robert	~~Alex~~	ABC Corporation	~~Gogo~~	Michael	81723

Ledger : XYZ Corporation - Partner

my_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Signature**

witness_ledger								
TransactionID	**From**	**Amount**	**To**	**Witness1**	**Partner1**	**Witness2**	**Partner2**	**Sign_reciever**
1003_1001	John	10	William	David	Beera	Robert	Gogo	42983
1002_1002	Robert	8	William	John	Beera	Gogo	ABC Corporation	21389

Functions

Random Number "Teleportation"

In the second step of a Teller transaction, a set of random numbers is used to randomly pick witness and Partner nodes. Witness nodes are picked from regular nodes pool, and Partner nodes are picked from a pool of Partner nodes. It is also important to point out

that senders and receivers are also regular nodes, but obviously, they cannot act as witness nodes for their own transaction. In a given transaction, any node can only act as one participant at a time.

The random number "teleportation" function is not only used to generate a random number, but also to make sure that the random number generated by node x at a time t is the exact same as the random number generated by node y and node z at time t. Therefore it is called random number teleportation; because nodes generate the same random number without transmitting them to each other.

So essentially, random numbers are time-bound and are the same for everyone at time X.

$RandomNumber_1 = F_x\ (Node^x\ Time^{t0}) = F_x\ (Node^y\ Time^{t0}) = F_x\ (Node^z\ Time^{t0})$
$RandomNumber_2 = F_x\ (Node^x\ Time^{t1}) = F_x\ (Node^y\ Time^{t1}) = F_x\ (Node^z\ Time^{t1})$
$RandomNumber_3 = F_x\ (Node^x\ Time^{t2}) \neq F_x\ (Node^y\ Time^{t2}) \neq F_x\ (Node^z\ Time^{t2})$

Pseudocode

```
FxRandomNumberTeleportation = function ()  {
                //Configured at predefined time by Partner nodes
                 Const frequency = 1 ; // seconds
                 Const time = now () ;
                Var jellyFishRandomNumber = FxGetJellyFishRandomNumberFromServer();
                //This array contains x random numbers valid for next one second.
                Return Array_Random_No[] =  FxGenerateRandomNumberOnNode  (
                jellyFishRandomNumber )
}
```

The random number teleportation function has two constants defined: Frequency & Time. Frequency is the amount(unit) of time (in seconds) for which these random numbers are valid and time is current timestamp.

Jellyfish random number (https://blockchainplus.co/randomnumber/) is a random number generated by observing jellyfish movements and downloading screenshots from jellyfish aquarium. The idea is to create "randomness" and "data liquidity" so that there is plenty of data created every second, which will help generate random numbers efficiently. Teller needs true/false random numbers. The first true random number is jellyfish random number which is absolute random number. The second one is false random number because even though it is generated from a true random number, it was re-generated by an algorithm with predefined constraints (refreshed at certain frequency by Partner nodes). The result is a true/false random number that perfectly mimics characteristic of a true random number i.e. a number that is impossible to predict and is generated on every node independent of each other.

Finally function FxGenerateRandomNumberOnNode generates a set of random numbers. This array is mapped against the unit of time and random numbers so that it can be used for the next t amount of time. Below is a representation of the output of this function.

Array = (
‘timeUnit^{0}’ => ‘randomnumber1’,
‘timeUnit^{1}’ => ‘randomnumber2’,
‘timeUnit^{2}’ => ‘randomnumber3’,
.....
‘timeUnit^{x}’ => ‘randomnumberx’,
)

For each timeunit^{n}, a random number is assigned in this array and it can be used by nodes for transaction operation. A sample is posted below for the above array definition.

Array = (
‘0’ => ‘12938719238712’,
‘1’ => ‘39187248237643’,
‘2’ => ‘39847237642837’,
.....
‘100,000’ => ‘39817239712833’,
)

It is important to note that each node may decide to generate a set of random numbers for all times and store it in their local storage. This may be useful when validating old transactions. Also, this function ensures that all nodes generate the same number versus a unit of time. It will help build integrity in function and transaction validation.

Snapshot Validation

Snapshots are used twice in the lifetime of transaction execution. First it is sent to corresponding partner node pairs after transaction is committed and second it is being checked by validating nodes before the start of next transaction. Both sender and receiver node snapshot needed to be checked. And as discussed before, snapshot nodes (four) are not assigned permanently with nodes, they rotate with defined algorithm and configuration data. A snapshot cannot be updated by its own node. E.g. Node A cannot update its own snapshot and send it to Node1, Node2, Node3, Node4. It should be done by any other node (other than Node A and snapshot nodes) and ideally by validating node selected at that point of time to perform transaction validation. If not that through that way, Node A will invalidate itself ledger.

Algorithm (Algo) Generation

An algo is generated on a predefined schedule; let's say, once a day. The algorithm is generated by all Partner nodes in consensus. The algorithm is the final piece of the function which is used by all the nodes to generate the "same" random number at the same time (See Random number "Teleportation"for more information). The algorithm contains a variation of formulas, variables, and functions to perform calculations. In its simplest definition, the function will look like the following.

Day 1:Algo^{Fx1} (String) = return FxPrettifyRandomNumber(HEX(String * Var^{1} + String * Var^{1}))
Day 2:Algo^{Fx1} (String) = return FxPrettifyRandomNumber(HEX(String * Var^{1} - String * Var^{1}))
Day 3:Algo^{Fx1} (String) = return FxPrettifyRandomNumber(HEX(String * Var^{1} * String * Var^{1}))

FxPrettifyRandomNumber

A function to finalize random numbers based on predefined parameters such as length, minimum number, and maximum number. A number is extracted based on startindex, which is typically 0, and predefined length. The number is then typecast from string to number and incremented until it fulfills the conditions. Below is the pseudocode.

```
FxPrettifyRandomNumber = function ( InitialNumber ) {
                    StartIndex = 0;
                    Length= X1;
                    Minimum = X2;
                    Maximum= X3;
                    finalRandomNumber =
                    finalRandomNumber = typeof finalRandomNumber  INT;
                    While ( finalRandomNumber < Maximum And finalRandomNumber > Maximum ){
                            finalRandomNumber ++;
                    }
                    Return finalRandomNumber
}
```

F_xConvertRandomNumberToNodeNumber

A function which converts a random number to node number. Below is the pseudocode.

```
fxConvertRandomNumberToNodeNumber = function ( randomNumber, isPartner = false ) {
                    Const NodeNumberStart =1;
                    Var NodeNumberEnd = config.regularNodeNumberEnd;
                    if(isPartner) {
                            NodeNumberEnd = config.PartnerNodeNumberEnd
                    }//endif
                    Return = NodeNumber = Fx (NodeNumberStart , NodeNumberEnd );
}
```

Sync Time

The SyncTime function is mainly used during boot time. It is important to sync times between all nodes so that they can perform the random number teleportation function effectively. Time can be stored as epoch unix time or a standard agreed reference to store in 64-bit integer data structure. The SyncTime process starts with a special sync_time network ping request initiated by the wallet registration node Node$_X$. A wallet registration node is defined as any regular or Partner node which is trying to

register itself to the network. It will be uncommon for a Partner node to perform frequent registration because of uptime requirements; however, for regular node registration, toggling is expected.

The sync_time ping request is then transmitted to 5 randomly selected nodes. This is a true random number generated by the node and not necessarily returned from the random number teleportation function. As there is no such requirement for this random number to be "teleported" to another nodes, there is no need use the random number teleportation function. In other words, no other node really needs to know which five nodes were chosen by $Node_X$, which is trying to be online.

The onus is on $Node_X$ to make sure that it syncs its time properly so that if a hacker node $Node_H$ tries to hack the network by choosing other hacker nodes $Node_{H+1}$ $Node_{H+2}$ $Node_{H+3}$$Node_{H+4}$ and $Node_{H+5}$ to populate its node with a fake time, it will automatically invalidate its future transactions, and may put the integrity of its existing my_ledger into question.

As there are two network types (transactional network and Partner network), this function is only responsible for randomly selecting nodes from the transactional network only and syncing time with them. This is sufficient, and there is no need to perform a separate sync time function with the Partner network. The main reason is that the transactional network consists of all the nodes which may act as a primary participant at some point in time. This includes Partner nodes as well.

Node Initiation Process

When a new node needs to be added to the network, regardless of its type (i.e.; whether it is a regular node or a Partner node), it must withstand the node initiation process. This process is different for different node types. The node initiation must be completed before starting the boot process for the first time a node is introduced to the network. The main reason why the node initiation process is needed and is kept separate from the boot process is because both regular and Partner nodes carry two types of transactional identification IDs. These transactional IDs are myID and loginID. myID is a permanent ID which is assigned to a node at the initiation process and kept with that node forever. It is a permanent ID number which is different from loginID in the sense that the latter is a temporary ID assigned during the boot process and only kept until the node is online. In addition, the above Partner network also carries a third type of identification which is named Partner ID as PartnerID.

The node initiation process is performed by calling the function F_XNetwork_Crawl_And_ Assign_Me place ('myID') and by setting 'IdType' parameter to 'myID'. This function is responsible for crawling the networks which are

set up in a binary tree or similar topology and assigning a place at the end. There are two main network types. One is the transactional network, which is a collection of both regular nodes and Partner nodes, and the other is the Partner network, which is only restricted to Partner nodes. These details are discussed in detail in a separate section. After a new node is initiated, it will go through the boot process in case it needs to be 'booted' at the same time. Please also note that the node initiation process does not need to perform the sync_time function, as there is no transaction happening, and the sync_time will eventually happen during the boot process. Finally, for every node a corresponding set of partner node is chosen. Snapshot nodes are assigned through a predefined but changing mathematical formula (algorithm + configuration data). These nodes will store snapshot of this node's my_ledger after any transaction (send or receive). The predefined formula is being updated by partner nodes during housekeeping to ensure predictability of collusion remains low or none (as discussed in separate section, we use POS for updating algorithm and configuration data).

Boot Process

A boot process is commonly defined as a startup process. In Teller's terminology, it refers to an attempt by a regular node Node_R to register its wallet with the network, and a Partner node Node_g to perform wallet registration, in addition to invoking the Partner certification process, which is required to register on the Partner network. Please note that the boot process is different than the "Node Initiation Process" in that the boot process is the node registration process for existing initiated node that turn their hardware on/off.

Boot Process for Regular Node

Sync Time

The first step in the boot process is to sync time with randomly selected nodes. As discussed in the Sync Time section, a true random number is used to select nodes from the transactional network to make sure that this node's time is the same as the ones in the network.

Wallet Registration

A wallet is any node which may act as a primary participant (sender or receiver) inside the transactional network. In fact, the transactional network only consists of such nodes, and no other nodes are allowed. Although, at time of writing, all nodes in the Teller network may act as a primary participant at some point in time, which makes them all eligible to act like a wallet. A wallet can thus be used to perform ledger transactions. Wallet registration can be defined as a process which is primarily responsible for bringing a node online. For a wallet to be registered on the transactional network, first it needs to ensure that this node Node_X has already gone through the initiation process.

This can simply be accomplished by checking to see if it has myID. myID identification is a permanent identification given to all nodes and used during the validation of transactions.

The final step is to perform F_XNetwork_Crawl_And_Assign_Me place ('loginID') by setting the 'IdType' parameter to 'loginID'. This automatically implies that a node needs to be registered in the transactional network and only assigned a temporary login ID. At the time of this writing, there are only two types of networks (i.e.; transactional network and Partner network) and nodes in both networks have myID and loginID set. In the Partner network, however, a third type of ID also needs to be set, which is PartnerID.

Once completed, a node will find its place in the respective network, whether it is the transactional network for regular and Partner nodes or the Partner network for Partner nodes exclusively.

Boot Process for Partner Node

Wallet Registration

The wallet registration process for a Partner node is exactly the same as a regular node, except for the fact that in the former, the case function to

F_XNetwork_Crawl_And_Assign_Me place is called twice instead of only once.
F_XNetwork_Crawl_And_Assign_Me place ('loginID')
F_XNetwork_Crawl_And_Assign_Me place ('PartnerID')

The first time when a network crawls and assigns, and a place function called, the 'loginID' parameter is passed to identify that this new Partner node $Node_G$ wants to register in the transactional network, and next time, the parameter 'PartnerID' is passed to specify its intent to be included in the Partner network. Two important points to consider here:

a) Inclusion in the transaction network is important because $Node_G$ may act as a transactional primary participant at any time.
b) Even though the above function will assign $Node_G$ its place in the Partner network, the network would not accept this Partner as it is, because it is still missing a Partner certificate.

Get and Install Partner Certificate

After the $Node_G$ registers itself in the Partner network by getting a Partner ID, it will be eligible to issue a get_certificate broadcast message to all Partner nodes. This

process ensures that the newly registered $Node_G$ is certified and can act as a Partner node. To do so, the network as a whole has to ensure that the $Node_G$ fulfills minimum computational (i.e.; software and hardware) and uptime requirements. This is mainly done by an application which triggers the network certification process.

First step to generate a certification for a new node is triggering the Partner Node Consensus Process. The F_X_Partner_Node_Consensus ('Partner_certification') function, when called with the parameter type='Partner_ceritication', returns a PartnerCertificateID. This certificate is then stored in the $Node_G$ database and will be used for validation purposes. Once this process is finished, $Node_G$ will find its place in both the transactional and Partner networks.

Get Latest Config, Random Number Teleportation Algorithm

Before performing any transactions, new Partner $Node_G$ (like any regular node) needs to update its local lightweight database with some important information. The $Node_G$ can also call function select random partner node function, which primarily generates a true random number within the range of Partner nodes IDs and returns an array of these Partner nodes as my ID. This data is then matched against all x numbers (typically five) returned by the Partner nodes to ensure its integrity and is subsequently used by the $Node_X$ for all its operations.

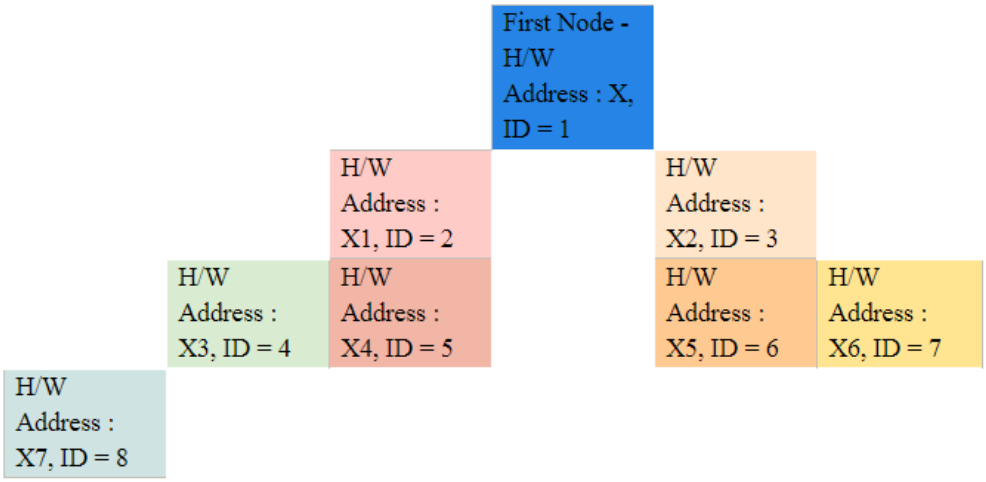
The above steps will now ensure that the $Node_G$ is visible on both the Partner and transactional networks, carries the proper identification, updates with the latest configuration/algorithm, and performs validation and transactional execution functions. In addition to the above, the $Node_G$ is now also capable of performing Partner housekeeping tasks.

Network Topology and Crawling

At the time of this writing, Teller topology is decided to be a binary tree; however, it can be changed by the Teller community according to its needs. As such, network crawling is accomplished through one of several binary tree traversal algorithms, such as depth-first search or divide and conquer, and can be done in any order, such as pre-order, in-order, or post-order. The main purpose of crawling optimization is to store and find data effectively in the network.

The figure below shows an example of nodes stored in the network with two different IDs. One of them is a permanent hardware address, and the other is a logical ID. In reference to the Teller network, there will be three IDs (myID, loginID, and PartnerID), where myID is a permanent ID (for example, the H/W address shown below) and

loginID will be a logical ID (for example, the ID shown below). A third ID, PartnerID, will also be a logical ID which is only stored for Partner nodes.



Partner Node Consensus Process

Partner nodes will need voting ability to perform housekeeping tasks, such as creating the random function teleportation algorithm, certifying new nodes in the network, assigning them rights so that they can act as Partner nodes, and generating configuration parameters. This activity must be done by every Partner node for network to achieve consensus.

The process is initiated by one Partner node by signing a blockchain message with its private key, and that blockchain is replicated across the network by appending to other blockchain records of neighborhood nodes. The process triggers multiple multi-directional blockchain queues, until all nodes reach consensus and sign the blockchain. This means that message may go back and forth until an agreement has been reached.

Node Identification Types

There are three types of node identification numbers: myID, loginID, and PartnerID. All nodes in the transactional network must contain myID and loginID. In addition nodes in the Partner network must contain PartnerID. The difference is stated below.
-myID: A permanent ID assigned to all type of nodes.
-loginID: A temporary ID assigned to all type of nodes.
This ID is assigned at the boot time and nullified when the device goes offline.
- PartnerID: A permanent ID assigned to all nodes in the Partner network. This ID is only assigned after the nodes are certified as Partner nodes. It is important to identify these nodes with a separate ID for their proper identification and for keeping their integrity intact.

Redundancy

Each partner is required to provide redundancy services for other nodes in the network. When a new node is certified and accepted inside partner network, two partner nodes are randomly selected for whom this node will be storing redundant data. New nodes will always be required to store their data throughout its lifetime. However, data store request will be initiated by the two partner nodes.

Essentially this means that each partner node is providing redundancy services for two other nodes at the same time. It also means that its data also being redundant at two other places. This redundancy operation ensures that there are always two nodes which will have data copies in case there is scheduled or non-scheduled downtime for that partner node. Additionally, the selected formula will be incremental as well as cyclic in nature to make sure it always provides redundancy to all of nodes.

Please also note that the exact semantics are currently being drilled down. One option is that a constant unchanging formula along with a variable set of configuration data will be used in conjunction of each other to make sure the redundant node selection can be achieved easily and instantly in case of new nodes popping up and old nodes dying frequently. This mapping does not need to be stored anywhere in the database but should be achievable through formula with the help of configuration set of data.

In the event of nodes dropping off permanently, those nodes will be brought back to the pool to match with new node to produce maximum redundancy which provide greater contingency in event one or two nodes are down at same time. The system always makes sure that there is third node to assist other nodes for the validation function and keep the network up and running.

We are currently exploring multiple different options to enhance the quality of the data redundancy environment.

Example: NodeA is being selected to provide redundancy to NodeB, NodeC.

Node A <-- Node B, Node C
Node B <-- Node C, Node D
Node C <-- Node D, Node E
Node D <-- Node E, Node A
Node E <-- Node A, Node B

Risk Assessment

The following scenario is shown to demonstrate risk assertiveness in the Teller network. If there are a 1000 Regular and 1000 Partner nodes and a hacker node (which is trying to create false transactions in order to create fraudulent activity), the chance of success for the hacker node is 1/41417124750 during any transaction time.
However, given the fact that:
(a) Teller will be open-source, meaning bugs and security breaches will be reported and fixed faster
(b) Teller will be operating on a global network, which means the number of nodes will be much higher;
(c) Teller Partner node community will be performing extensive tasks regularly, such as bad node reporting, random algo generation, and certification issuing/revoking. This will further reduce security risks to a minimum.

Assumed Minimum BlockChain+ Configuration	
Regular Nodes	1000
Partner Nodes	1000

Legend	**Explanation**
A	Still high because W1 and W2 has no-reply and G1 and G2 has a match, so you still need 1000*1000*1000*1000 for 1000:1000 (Witness/Partner) node configuration
B	Issue will be reported to the Partner nodes as Partner should always reply
C	Enough confirmation that it is good
D	Considering a regular node has a 50% chance of staying up - meaning a regular node does not reply 50% of the time.

A True result indicates that the validation process was being validated and the participating nodes can now move forward to the transactional completion step.

	Nodes					
Result (Below)	Warrantor 1	Warrantor 2	Partner 1	Partner 2	Legend Reference	Odds (Based on assumed nodes configuration)
TRUE	Match	Match	Match	Match		41417124750
FALSE	Match	Match	Match	No Match		
FALSE	Match	Match	Match	No Reply	B	
FALSE	Match	Match	No Match	Match		
FALSE	Match	Match	No Match	No Match		
FALSE	Match	Match	No Match	No Reply	B	
FALSE	Match	Match	No Reply	Match	B	
FALSE	Match	Match	No Reply	No Match	B	
FALSE	Match	Match	No Reply	No Reply	B	
FALSE	Match	No Match	Match	Match		
FALSE	Match	No Match	Match	No Match		
FALSE	Match	No Match	Match	No Reply	B	
FALSE	Match	No Match	No Match	Match		
FALSE	Match	No Match	No Match	No Match		
FALSE	Match	No Match	No Match	No Reply	B	
FALSE	Match	No Match	No Reply	Match	B	
FALSE	Match	No Match	No Reply	No Match	B	
FALSE	Match	No Match	No Reply	No Reply	B	
TRUE	Match	No Reply	Match	Match	C,D	>166168000
FALSE	Match	No Reply	Match	No Match		
FALSE	Match	No Reply	Match	No Reply	B	
FALSE	Match	No Reply	No Match	Match		
FALSE	Match	No Reply	No Match	No Reply	B	
FALSE	Match	No Reply	No Reply	Match	B	
FALSE	Match	No Reply	No Reply	No Match	B	
FALSE	Match	No Reply	No Reply	No Reply	B	
FALSE	No Match	Match	Match	Match		
FALSE	No Match	Match	Match	No Match		
FALSE	No Match	Match	Match	No Reply	B	
FALSE	No Match	Match	No Match	Match		
FALSE	No Match	Match	No Match	No Match		
FALSE	No Match	Match	No Match	No Reply	B	
FALSE	No Match	Match	No Reply	Match	B	
FALSE	No Match	Match	No Reply	No Match	B	
FALSE	No Match	Match	No Reply	No Reply	B	

FALSE	No Match	No Match	Match	Match		
FALSE	No Match	No Match	Match	No Match		
FALSE	No Match	No Match	Match	No Reply	B	
FALSE	No Match	No Match	No Match	Match		
FALSE	No Match	No Match	No Match	No Match		
FALSE	No Match	No Match	No Match	No Reply	B	
FALSE	No Match	No Match	No Reply	Match	B	
FALSE	No Match	No Match	No Reply	No Match	B	
FALSE	No Match	No Match	No Reply	No Reply	B	
FALSE	No Match	No Reply	Match	Match		
FALSE	No Match	No Reply	Match	No Match		
FALSE	No Match	No Reply	Match	No Reply	B	
FALSE	No Match	No Reply	No Match	Match		
FALSE	No Match	No Reply	No Match	No Match		
FALSE	No Match	No Reply	No Match	No Reply	B	
FALSE	No Match	No Reply	No Reply	Match	B	
FALSE	No Match	No Reply	No Reply	No Match	B	
FALSE	No Match	No Reply	No Reply	No Reply	B	
TRUE	No Reply	Match	Match	Match	C	>166168000
FALSE	No Reply	Match	Match	No Match		
FALSE	No Reply	Match	Match	No Reply	B	
FALSE	No Reply	Match	No Match	Match		
FALSE	No Reply	Match	No Match	No Match		
FALSE	No Reply	Match	No Match	No Reply	B	
FALSE	No Reply	Match	No Reply	Match	B	
FALSE	No Reply	Match	No Reply	No Match	B	
FALSE	No Reply	Match	No Reply	No Reply	B	
FALSE	No Reply	No Match	Match	Match		
FALSE	No Reply	No Match	Match	No Match		
FALSE	No Reply	No Match	Match	No Reply	B	
FALSE	No Reply	No Match	No Match	Match		
FALSE	No Reply	No Match	No Match	No Match		
FALSE	No Reply	No Match	No Match	No Reply	B	
FALSE	No Reply	No Match	No Reply	Match	B	
FALSE	No Reply	No Match	No Reply	No Match	B	
FALSE	No Reply	No Match	No Reply	No Reply	B	
TRUE	No Reply	No Reply	Match	Match	C,A	>999000
TRUE	No Reply	No Reply	Match	Match	C,A	>999000

FALSE	No Reply	No Reply	Match	No Match		
FALSE	No Reply	No Reply	Match	No Reply	B	
FALSE	No Reply	No Reply	No Match	Match		
FALSE	No Reply	No Reply	No Match	No Match		
FALSE	No Reply	No Reply	No Match	No Reply	B	
FALSE	No Reply	No Reply	No Reply	Match	B	
FALSE	No Reply	No Reply	No Reply	No Match	B	
FALSE	No Reply	No Reply	No Reply	No Reply	B	

Derivation

Below shown r objects from a set of n objects. Here r is combination (2, 3 or 4) and objects are number of maximum of nodes (1000) in that particular network

${}_nC_r = n!/r!*(n - r)! = (n(n-1)(n-2)...(n-r+1))/r!$

Please note that ${}_nC_r = {}_nP_r/n!$ where ${}_nP_r$ is the formula for permutations of n objects taken r at a time.

Scenario one where all four nodes are required to return True, the formula will be

$1000!/4!(1000-4)! = 41417124750$

Seconds and third scenario we have

$1000!/3!(1000-3)! + 1000!/1!(1000-1)! = 166168000$

Fourth Scenario

$1000!/2!(1000-2)! + 1000!(2!(1000-2)! = 999000$

Even though odd of 999,000 looks very low the actual odds will be quite higher because above calculation does not consider 3 return values as described in below paragraph. Additionally, in Teller we have opportunity to set minimum node configuration operating requirements. This will increase the odds exponentially.

Please also note that nodes can reply FALSE, TRUE and No-Reply so in that case the odds will be even higher than above. Above odds are only minimum possible odds.

Conclusion

This research paper exploits an opportunity to implement private transactions in public, open-source blockchain architecture without compromising security, transparency, or trust. It effectively tackles problems of speed, bandwidth and privacy. Teller offers a solution to these problems with an open-source, ledger-based approach. Information is only stored in participating nodes. In the future, there will be a strong focus on operational aspects, such as quicker validation and execution, to increase Teller's use cases.

References

- The godfather whitepaper - Bitcoin: A Peer-to-Peer Electronic Cash System : Satoshi Nakamoto : https://bitcoin.org/bitcoin.pdf
- A Next-Generation Smart Contract and Decentralized Application Platform : https://github.com/ethereum/wiki/wiki/White-Paper
- Hyperledger – Blockchain Technologies for Business : hyperledger.org

Teller

Automate Trust

BLOCKCHAIN+

blockchainplus.co