The Generation Of Layer 1+ Blockchain

Ing. Robert Michálek

Marek Szotkowski

Vladimír Lieger

https://busyxchain.network

February 2024, v1.6

Abstract

With the rapid growth of blockchain technology, many new possibilities are opening up that seemed previously unsolvable. Key features such as transparency (permissionless, no third parties, trustless) and decentralization unlock new opportunities to solve challenges in nearly all public or private sectors. BusyXChain is a Layer 1+ blockchain network dedicated to bringing the advantages of public, private and hybrid chains into a single ecosystem and creating a layer of interconnectivity between anyone within the ecosystem, while keeping data confidential and secure. All in one L1+ blockchain, no bridges, and no cross chains.

Contents

1. Introduction

Blockchain is a much-discussed topic these days, and companies (and organizations, institutions and governments) have started to implement blockchain technology into their business processes, whether private or public. BusyXChain is a new eco-friendly blockchain technology that connects public and private organizations, individuals, DeFi and DAO projects, NFT and Metaverse projects, as well as any other type of company or project. BusyXChain also provides complete privacy and security for companies that prefer to implement a fully private network.

Our solution harnesses the advantages while removing the disadvantages of public, private and hybrid blockchains, all at once, allowing for interconnectivity between multiple parties, any project with any company. For example, DAO and DeFi projects can easily connect their layers into a L1 solution and run on the BusyXChain without the need for any external, risky infrastructure (for example bridges).

A complete, all-in-one L1 blockchain with unlimited use cases, from single users and small projects to global corporations and governments.

For individuals or small entities, the focus is channel privacy control, smart contracts, easy and straightforward token creation with utility in a broad spectrum, including DeFi, NFT, GameFi and Metaverse directly via the UI of our desktop crypto wallet. For larger entities, the focus is for our BusyXChain solution to enable interconnectivity of company databases in revolutionary new ways, one that was not possible until now. The solution that allows private and public layers (virtual layers on layer 1) to connect seamlessly without compromising privacy or reliability. Allowing for a smooth connection between multiple entities using different infrastructure, for example, connect private companies with DeFi, NFT and Metaverse all in one layer without unwanted interference and without compromising privacy or security of any of the individual channels.

1.1. Blockchain

A blockchain is a distributed database of records. It can be public (open to all) or private. The database comprises data blocks connected to the previous block, forming a chain. As well as the data itself, each block also contains a record of when the block was created or edited, making it very useful for maintaining a complex system of records that cannot be corrupted or lost.

Blockchain technology eliminates the issues of security and trust in several ways. First, new blocks are always stored linearly and chronologically. This means they are continually added to the “end” of the blockchain. Once the block is chained, it is very difficult to go back and alter the contents of the block. That is because each block contains its own hash, along with the hash and time stamp of the previous block. Hash codes are created by a math function that turns digital information into a string of numbers and letters. If that information is edited in any way, the hash code also changes.

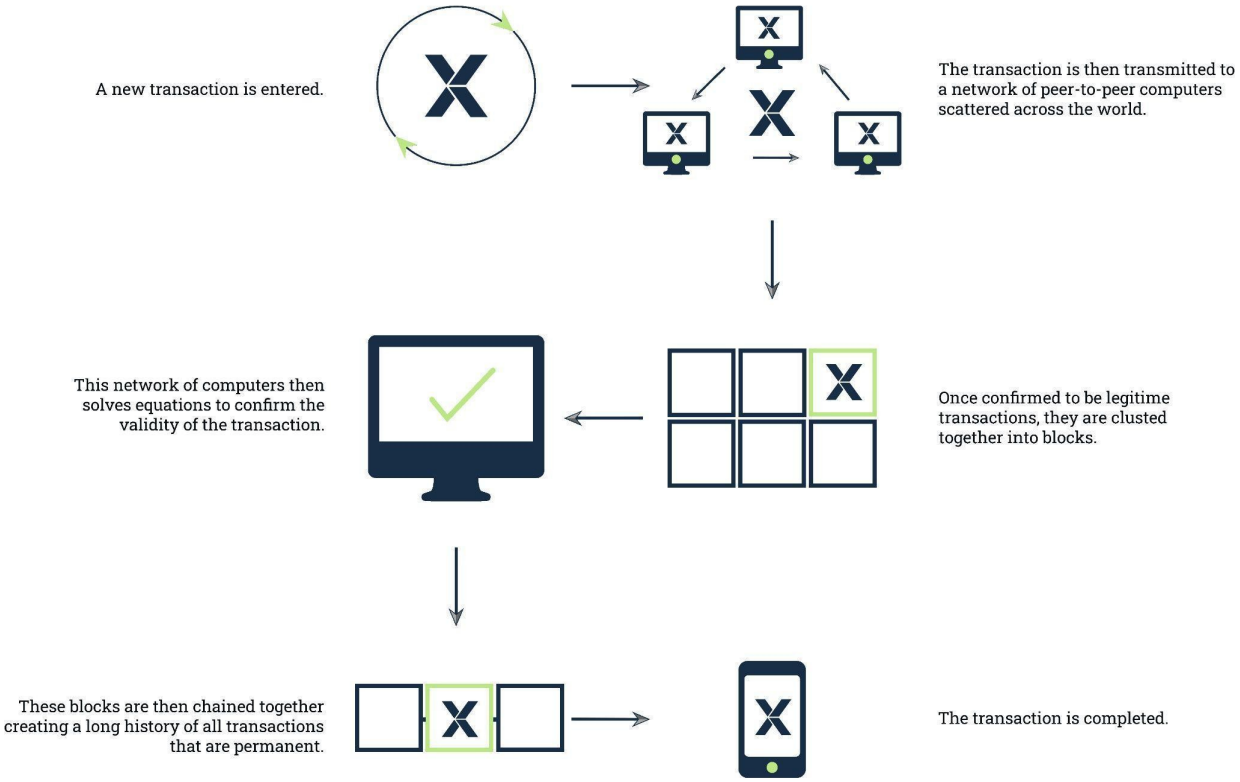

Figure 1: How BXCH transaction works in general

Users can view the entire blockchain as the blockchain database is duplicated across all network nodes. Transactions and data are processed not by one central administrator but by network users who work to verify the data and achieve a consensus. The BusyXChain solution was inspired by the original proof of stake consensus, and uses this component as a model in the network, a way of upgrading and restructuring the network for a fair and decentralized way to reach consensus, making it a completely unique solution.

1.1.1. Public (permissionless)

A public blockchain network is a blockchain network where anyone can join whenever they want. Essentially, there are no restrictions when it comes to participation. More so, anyone can see the ledger and participate in the consensus process. However, it can be problematic when you try to incorporate a public blockchain network with an enterprise blockchain. Public blockchains do

come with their fair share of flaws as well. In reality, due to the comprehensive governance process and network verification, they are often slower than private networks. Furthermore, public blockchains can attract dishonest people who may use the platform for malicious activities due to the anonymity of users on a public blockchain. Currently, there is no mechanism in the industry to completely eliminate this risk without compromising decentralization.

1.1.2. Private (permissioned)

A private blockchain is when only a single organization has authority over the network. Meaning it’s not open to the public. Private Blockchains are primarily used internally within an organization or company.

1.1.3. Hybrid

A hybrid blockchain is best defined as a blockchain that uses the essential components of both private and public blockchain solutions.

A hybrid blockchain is not open to everyone but still offers core blockchain features such as integrity, transparency, and security.

Hybrid blockchains are very customizable. Companies can decide who can participate in the blockchain or which transactions are made public. This theoretically brings the best of both worlds and ensures that a company can work with its stakeholders in the best possible way.

1.1.4. Consortium blockchain

The fourth type of blockchain, consortium blockchain, also known as a federated blockchain, combines public and private blockchains and contains centralized and decentralized features. Essentially, a consortium blockchain is a private blockchain limiting access to a particular group. By a group controlling a blockchain, one eliminates the risks that come with a single entity controlling the network on a private blockchain.

One can control consensus procedures in a consortium blockchain by so-called preset nodes. A validator node initiates, receives and validates transactions. Other nodes, called member nodes, can receive or initiate transactions. These blockchains enjoy more decentralization than private blockchains and therefore have higher levels of security. Although setting up consortium blockchains can be a difficult and tedious process, as it requires coordination between several organizations, which consequently emphasizes logistical challenges. Consortium blockchains can be compromised if a member node is breached. So the blockchain's regulations can be an obstacle to the network's functionality.

Banking services and payment systems are common uses for this type of blockchain. Different banks can form a consortium, deciding which nodes will be the validator nodes and validating the transactions. Research organizations can

create a similar model or organizations that want to track food. Consortium blockchains are an ideal solution for companies with a large supply chain, especially in the food, pharmaceutical and other industries with sourced materials.

2. Problems with other blockchains

There are currently a few blockchains that offer the possibility of creating public, private or hybrid networks. Most of them do not offer an option to use a combination of those. Another disadvantage is that building on any of the currently available protocols, in most cases, you would have to use the native programming language of the platform, which can be a hindrance. Furthermore, there is also the need for a connection between individual chains/projects. These connections tend to be very complex and can be vulnerable. In the latest months, there were many attacks focused on bridges (connection points between individual chains/projects).

BusyXChain utilizes all the benefits from the various types of blockchains while eliminating the disadvantages. Giving our users the freedom to develop the best solution for their unique requirements.

2.1. Public/Private blockchains

Public chains have advantages in independence, transparency and trust; disadvantages are performance, scalability and security. Considering the above, good use cases might be cryptocurrency or document validation.

Private chains are advantages of access control and performance; the disadvantages are transparency, audibility and centralized control. This is why use cases often cover supply chains and asset ownership. For the most part, private and public blockchains exist as separate entities, users and organizations have to choose one that works best for them. BusyXChain challenges the status quo by creating an ecosystem that incorporates all blockchain types, public, private, hybrid and consortium. This allows users to develop a solution using any type of blockchain and then creating connections to other projects. Our solution allows users to customize privacy policies based on their requirements.

An example use case for each blockchain type: Let’s say we have a car manufacturer, and they need to communicate with each of their dealerships electronically, exchanging orders, parts, etc.

Public blockchain: Everyone can join the public blockchain, which is definitely not ideal as important company information becomes publicly accessible. Part numbers, wholesale prices, unit sales, etc. will all become public knowledge.

Private blockchain: In this case, a private blockchain is much better than a public blockchain but would still be highly limited. It won’t be accessible to new dealerships unless specific access is granted, and dealership staff members may not have full access to important information. Dealerships will need to manage user’s rights for each of their staff members. Furthermore, dealerships may have

separate stock allocations and pricing models based on their moving volumes. If all dealerships have access to this information, it could be challenging for the car manufacturer to negotiate better deals for different dealerships.

Hybrid blockchain: The best match! The car manufacturer can have a private channel with all dealership details, and each dealership has a separate channel that will interact with the private channel with permission. The dealership's channel can be configured easily; for example, SellerOneChannel can communicate with channel CarManufacturer only with restricted access. For example, they would only be able to see the manufacturer's wholesale price for models of cars that a particular dealership sells. This hybrid chaincode can run on an already scaled and decentralized blockchain in the channel, the company can define who will be able to see the data.

Consortium blockchain: This network type is unsuitable for the mentioned example. A Consortium contains several organizations that join the network. According to set rules, for example, the majority of peers must approve most things on the network. From this perspective, it doesn't make sense to own/share a network with other sellers, and decentralizing the network is not a priority either.

2.2. Bridges

Blockchain bridges work just like the bridges we know in the physical world – a blockchain bridge connects two blockchain ecosystems. Bridges facilitate communication between blockchains through the transfer of information and assets. All blockchains developed in isolated environments cannot natively communicate, and tokens cannot move freely between blockchains.

Bridges have become extremely risky over the recent months. There have been plenty of crypto attacks directly to the bridges and hundreds of millions of dollars have been stolen. The problem with bridges is that there are multiple points of failure, from a security perspective, like smart contract vulnerabilities, technology vulnerabilities, censorship and custodial vulnerabilities.

BusyXChain does not require any form of bridges or cross chain integration. Thanks to channels and permissions. Data from an individual channel is not accessible to the other channels. If there is a vulnerability in the smart contract running on a channel, only the particular channel would be disrupted, nothing else. It would always be the responsibility of the channel's owner to secure their application/smart contract. BusyXChain contains the vulnerability and protects other channels so that they cannot be exploited and attacked due to a single channel's vulnerability.

2.3. Difficult custom token creation

With the massive boom in token creation over the recent years has – mainly NFT, GameFi, DeFi, and Metaverse. Thousands of projects were built on different blockchains. As mentioned previously, bridges were required to allow for cross chain interconnectivity. Furthermore, creating tokens with specific functions is difficult for the average user and token development is a costly affair. BusyXChain provides an easy-to-use solution for quick custom token creation via our integrated Token Creator feature in our Desktop Wallet. Anyone can create a custom token and connect it with another project/company/user seamlessly and enjoy all benefits of blockchain technology.

3. BusyXChain solution

BusyXChain is a blockchain solution for developing and connecting blockchain-based products, solutions, and applications using plug-and-play components. It is based on modular architecture that delivers high degrees of flexibility, resiliency, confidentiality and scalability. This enables solutions developed with BusyXChain to be adapted for just about any industry.

Thanks to the modularity of our blockchain, scalability, such as sharding, can be achieved. Each new solution contributes to the decentralization of the entire network. This way, BusyXChain can process even more blocks at the same time.

BusyXChain has an additional virtual layer above the core blockchain network. The virtual layer has its own access rules and employs its own mechanism for transaction ordering, providing an additional layer of access control. This is beneficial when users want to limit data exposure and maintain privacy. Ensuring that it can be viewed by the relevant parties only. For example, when two competitors are on the same network – the virtual layer offers private data collection and accessibility, where sensitive information will not be exposed to the competitor.

3.1. Why BusyXChain?

BusyXchain is a Layer 1+ modular blockchain network with the ability to operate on all types of blockchains available today. Users can run public, private, hybrid or consortium blockchains. Each type of blockchain is suitable for different use cases. In addition, users can also use the BusyXchain network to create smart contracts, create tokens, and develop various platforms and applications. Users have the option to create a fully customized solution on the BusyXChain network. This level of functionality and freedom ensures that BusyXChain has unlimited use cases and can be used by anyone to build just about anything.

The significant advantage is that multiple organizations can coexist in a single ecosystem. For example, a consortium is formed, and the consortium participates in network management and data sharing according to a predefined set of rules.

Each connected organization within the consortium, with network elements (Peer, Orderer) becomes part of the entire BusyXchain network while simultaneously remaining independent of the BusyXchain network. The consortium can also create a so-called shard and thus scale the network as a whole.

Any project or company can join with their existing infrastructure (for example, ERP) and move the whole business or just particular processes to the blockchain. Our network uses classic programming languages like Java, Golang, and NodeJS, so developers do not need to learn a new language in order to develop on BusyXChain.

Another advantage is that communication between projects created on BusyXchain is rather simple. Permissions can be set up using a “channel”. No complex solutions such as bridges or cross-chains are needed. BusyXchain allows multiple projects to operate in a single ecosystem while remaining completely independent but still allowing for complete interconnectivity.

3.2. Technical details

The BusyXChain is based on the Hyperledger Fabric framework that provides a modular and easily expandable layer-1 solution. The essential components are Peers and Orderers. We expanded the network with the possibility of connection with superficial nodes such as physical servers, virtual private servers, Kubernetes clusters and cloud solutions. That is how the existing infrastructure with a blockchain network can be extended without creating a new, separate blockchain.

3.2.1. Network architecture

The BusyXChain network consists of Peers, Orderers, CA servers, APIs and Events.

Each Peer and Orderer:

- Has an external endpoint for communication with other parts of the network. This means that it doesn't matter if you connect, for example, a VPS server or a cluster to the network – everything behaves the same. The communication between Peers and Orderers uses GRPC and Gossip protocol.
- Stores ledgers (this depends on how many channels are running on each Peer).

Each Peer works as a Committing Peer, some are Endorsement Peers (these are the Peers where chaincode is installed and processed). Another type of Peer is an

Anchor Peer. Anchor Peers have to be defined for other companies (organizations) joining BusyXChain – it will be a Peer Node on the specified channel that all other Peers can discover and communicate with. Peers are using the Gossip protocol to broadcast their existence in the network and to different organizations.

With the use of APIs and Events, it is possible to communicate with the entire network.

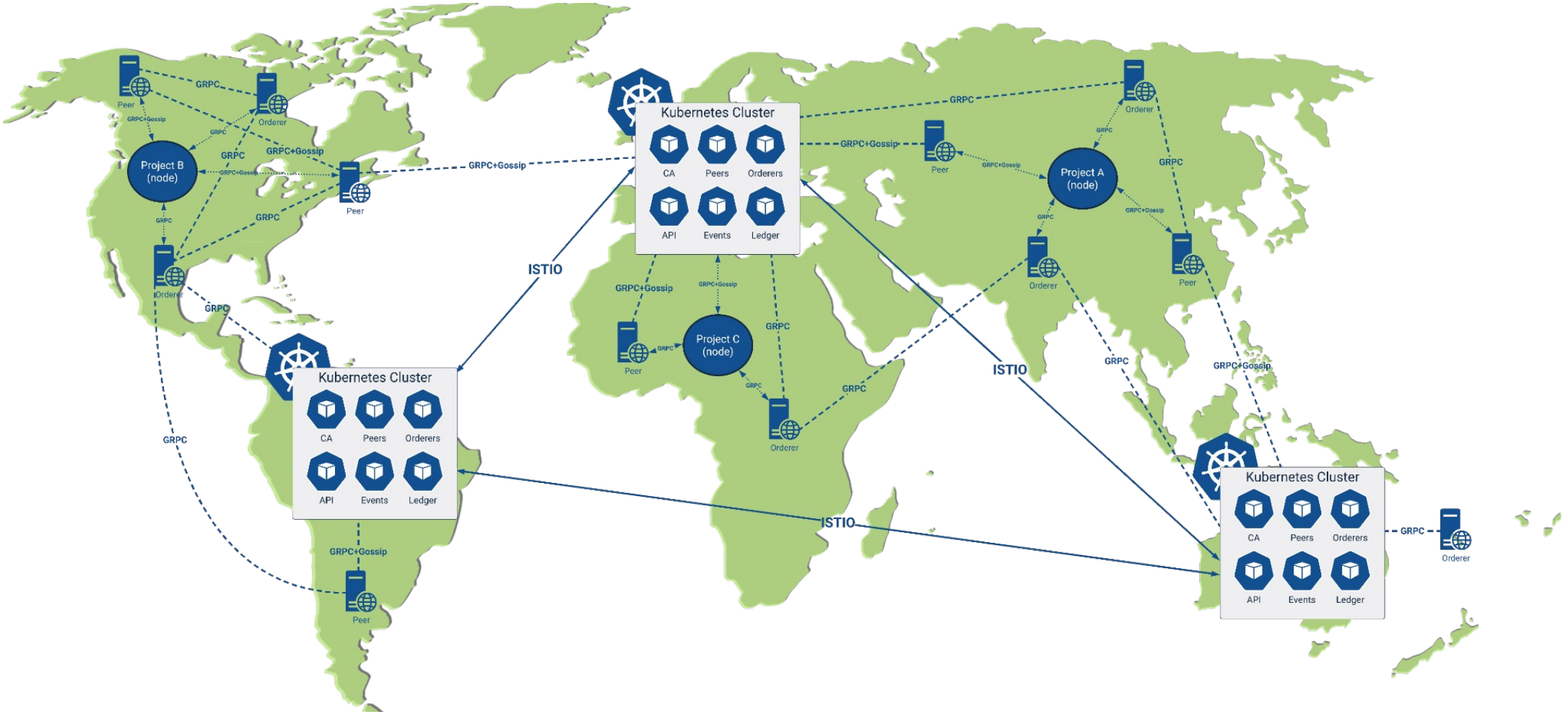

Figure 2: BusyXChain Network Channels Example

3.2.2. Gossip protocol

Peers leverage the Gossip protocol to broadcast ledger and channel data in a scalable fashion. Gossip messaging is continuous, and each Peer on a channel constantly receives current and consistent ledger data from multiple Peers. Each gossiped message is signed, thereby allowing Byzantine participants who send faked messages to be easily identified and the distribution of the message(s) to unwanted targets to be prevented. Peers affected by delays, network partitions, or other causes resulting in missed blocks will eventually be synced up to the current ledger state by contacting peers possessing these missing blocks.

Gossip messaging

Online Peers indicate their availability by continually broadcasting "alive" messages containing the public key infrastructure (PKI) ID and the sender's signature over the message. Peers maintain channel membership by collecting these alive messages; if no Peer receives the alive message from a specific Peer, this "dead" Peer is eventually purged from channel membership. Because "alive" messages are cryptographically signed, malicious Peers can never impersonate other Peers, as they lack a signing key authorized by a root certificate authority (CA).

In addition to the automatic forwarding of received messages, a state reconciliation process synchronizes world states across Peers on each channel. Each Peer continually pulls blocks from other Peers on the channel to repair its own state if discrepancies are identified. Because fixed connectivity is not required to maintain gossip-based data dissemination, the process reliably provides data consistency and integrity to the shared ledger, including tolerance for node crashes.

As the channels are segregated, Peers on one channel cannot message or share information on any other channel. Though any Peer can belong to multiple channels, partitioned messaging prevents blocks from being disseminated to Peers that are not in the channel by applying message routing policies based on Peers' channel subscriptions.

Gossip security

The Peer’s TLS layer handles the security of point-to-point messages and does not require signatures. Peers are authenticated by their certificates, which are assigned by the CA. Although TLS certs are also used, the Peer certificates are authenticated in the gossip layer. Ledger blocks are signed by the ordering service and then delivered to the leader Peers on a channel.

Authentication is governed by the membership service provider for the Peer. When the Peer connects to the channel for the first time, the TLS session binds

with the membership identity. This authenticates each Peer to the Connecting Peer concerning membership in the network and channel.

3.2.3. Transaction flow

The blocks of transactions are “delivered” to all Peers on the channel. The transactions within the block are validated to ensure endorsement policy is fulfilled and that there have been no changes to the ledger state for reading set variables since the read set was generated by the transaction execution. Transactions in the block are tagged as valid or invalid. For the full transaction flow with details, check the official Hyperledger documentation: https://hyperledger-fabric.readthedocs.io/en/latest/txflow.html.

3.2.4. RAFT consensus

BusyXChain blockchain uses RAFT consensus. The RAFT consensus states that each node in a replicated state machine (server/node) can stay in any of the three states: Leader, Candidate, and Follower. The image below provides the necessary visual aid.

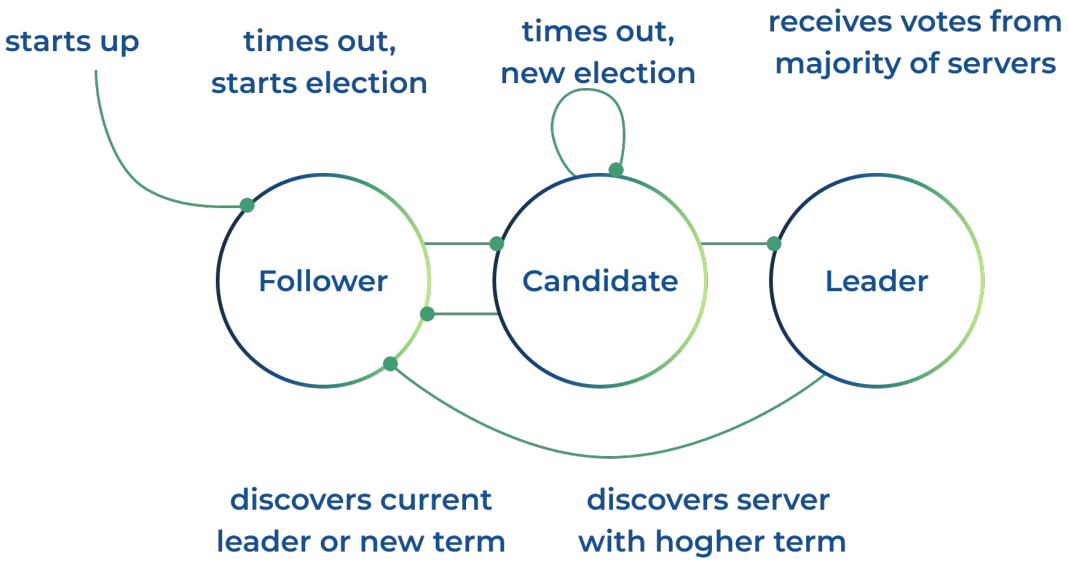

Figure 3: RAFT consensus – Follow, Candidate, and Leader

Under normal conditions, a node can stay in any of the above three states. Only a Leader node can interact with the client; any request to the Follower node is redirected to the Leader node. A Candidate node can ask for votes to become the Leader node. A Follower node only responds to the Candidate node(s) or the Leader node.

The RAFT algorithm divides time into small terms of arbitrary length to maintain these server statuses. Each term is identified by a monotonically increasing number, called Term Number.

Term Number

The Term Number is maintained by every node and is passed during communications between nodes. Every Term Number starts with an election to determine the new Leader. The Candidates ask for votes from other server nodes (Followers) to gather the majority. The candidate becomes the current Term Leader if the majority is collected. If no majority is established, the situation is called a split vote, and the Term ends with no Leader. Hence, a term can have at most one Leader.

Purpose of maintaining Term Number

The following tasks are executed by observing the Term Number of each node:

- Servers update their Term Number if their Term Number is less than the Term Numbers of other servers in the cluster. When a new term starts, the Term Numbers are tallied with the Leader or the Candidate and updated to match the latest one (Leader's).
- Servers update their Term Number if their Term Number is less than the Term Numbers of other servers in the cluster. When a new term starts, the Term Numbers are tallied with the Leader or the Candidate and updated to match the latest one (Leader's).

As was written earlier, the Term Number of the servers is also communicated. If a request is achieved with a stale Term Number, the said request is rejected. This means a server node will not accept requests from the server with a lower Term Number.

Remote Procedure Calls

The RAFT algorithm uses two types of Remote Procedure Calls (RPCs) to carry out the functions:

- "RequestVotes" RPC call is sent by the Candidate nodes to gather votes during an election.
- "AppendEntries" RPC call is used by the Leader node to replicate the log entries and as a heartbeat mechanism to check if a server is still up. If the heartbeat is responded to, the server is up; the server is down. Be noted that the heartbeats do not contain any log entries.

Leader election

To maintain authority as a Leader of the cluster, the Leader node sends a heartbeat to express dominion to other Follower nodes. A leader election takes place when a Follower node times out while waiting for a heartbeat from the Leader node. At this point, the timed-out node changes its state to the Candidate

state, votes for itself and issues “RequestVotes” RPC[1] to establish a majority and attempt to become the Leader. The election can go the following three ways:

1. The Candidate node becomes the Leader by receiving the majority of votes from the cluster nodes. At this point, it updates its status to Leader and starts sending heartbeats to notify other servers of the new Leader.
2. The Candidate node fails to receive the majority of votes in the election, and hence the term ends with no Leader. The Candidate node returns to the Follower state.
3. If the term number of the Candidate node requesting the votes is less than other Candidate nodes in the cluster, the “AppendEntries” RPC[2] is rejected, and other nodes retain their Candidate status. If the Term Number is greater, the Candidate node is elected as the new Leader.

[1] “RequestVotes” RPC is sent by the Candidate nodes to gather votes during an election.

[2] “AppendEntries” RPC is used by the Leader node to replicate the log entries and, as a heartbeat, a mechanism to check if a server is still up. If the heartbeat is responded to, the server is up; the server is down. Be noted that the heartbeats do not contain any log entries.

Figure 4: Leader election

Log replication

Each request made by the client is stored in the Logs of the Leader. This log is then replicated to other nodes (Followers). Typically, a log entry contains the following three pieces of information:

- The command specified by the client to execute.
- Index to identify the entry's position in the node log. The index is 1-based (starts from 1).
- Term Number to ascertain the time of the entry of the command.

The Leader node fires "AppendEntries" RPCs to all other servers (Followers) to sync/match up their logs with the current Leader. The Leader keeps sending the RPCs until all the Followers safely replicate the new entry in their logs.

There is a concept of entry commit in the algorithm. When the majority of the servers in the cluster successfully copy the new entries in their logs, it is considered committed. At this point, the Leader also commits the entry in its log to show that it has been successfully replicated. All the previous entries in the log are also considered committed for apparent reasons. After the entry is

committed, the Leader executes the entry and responds back with the result to the client.

It should be noted that these entries are executed in the order they are received.

If two entries in different logs (Leaders and Followers) have identical index and term, they are guaranteed to store the same command, and the logs are identical up to that point (Index).

Safety

To maintain consistency and the same set of server nodes, it is ensured by the RAFT consensus algorithm that the Leader will have all the entries from the previous terms committed in its log.

During a Leader election, the “RequestVote” RPC also contains information about the Candidate’s log (like Term Number) to figure out which one is the latest. If the Candidate requesting the vote has less updated data than the Follower from which it is requesting the vote, the Follower simply doesn't vote for the said Candidate. The following excerpt from the original RAFT paper clears it similarly and profoundly.

RAFT determines which of the two logs is more up-to-date by comparing the index and term of the last log entries. If the logs have previous entries with different terms, then the log with the later term is more up-to-date. If the logs end with the same term, then whichever log is longer is more up-to-date.

Rules for safety in the RAFT protocol

The RAFT protocol guarantees the following safety against consensus malfunction by its design:

- Any node in the network can become the Leader. So, it has a certain degree of fairness.
- Leader election safety – At most, one Leader per term.
- Log matching safety – If multiple logs have an entry with the same index and term, those logs are guaranteed to be identical in all entries up through to the given index.
- Leader completeness – The log entries committed in a given term will always appear in the logs of the Leaders following the said term.
- State Machine safety – If a server has applied a particular log entry to its state machine, then no other server in the server cluster can use a different command for the same log.
- The Leader is Append-only – A Leader node (server) can only append (no other operations like overwrite, delete, or update is permitted) new commands to its log.

- Follower node crash – When the follower node crashes, all the requests sent to the crashed node are ignored. Further, the crashed node can't participate in the Leader election for obvious reasons. When the node restarts, it syncs its log with the leader node.

CFT (Crash Fault Tolerant)

Because the system can sustain the loss of nodes, including Leader nodes, as long as most ordering nodes remain, RAFT is "crash fault-tolerant" (CFT).

The blockchain network consists of a series of nodes arranged from 0 to n - 1, where n is the number of nodes in the network. There is a so-called maximum number of "bad" nodes that the network can tolerate. If this number of "bad" nodes, called the constant f, is not exceeded, the network will function properly. The constant f is equal to one-third of the nodes in the network. At least 2/3 of the nodes must be functional and honest for the algorithm to work.

$$n = \text{Total \# of nodes in network}$$
$$f = \frac{n - 1}{3} \quad \text{(Max \# of faulty nodes)}$$

Figure 5: Equation for 1 constant calculation

In other words, if there are three nodes in a channel, it can withstand the loss of one node (leaving two remaining). If there are five nodes in a channel, it can lose two (leaving three remaining nodes). This feature of a RAFT is a factor in the establishment of a high-availability strategy.

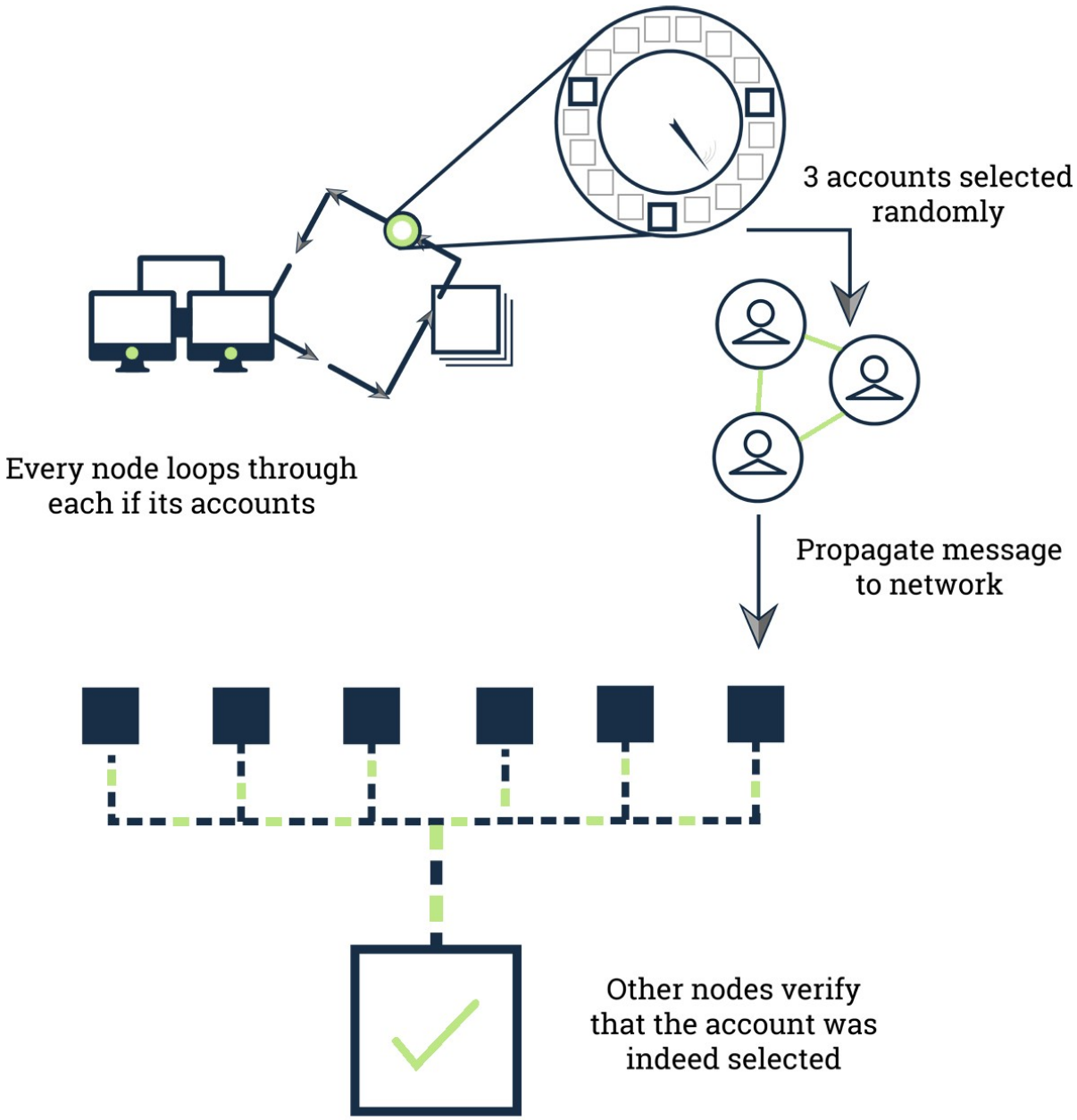

Figure 6: BusyXChain – transaction verification

3.2.5. Staking

BusyXChain staking model is similar to pure Proof of Stake (PoS) with a few specific improvements. Based on Committing Peers, every Peer in the BusyXChain network will work like Committing Peer. This means it does not matter how many coins the user has; each Peer can be chosen as a Committer Peer. This ensures the parity of all nodes in the network. The staking reward is set up for 11% p.a.[3]

[3] E.g.: Staking reward is set to 10.7–11.2% p.a. Staking reward is affected by the holding time of the coins, i.e., the time for which the user holds the coins. Each year does not have exactly the same time periods, so staking reward can vary within the given deviations.

3.3. Channels

Channels are a private method of “subnet” communication between organizations. As a result, most changes to the channel configuration need to be agreed to by other channel members. A channel would not be helpful if an organization could join the channel and read the data on the ledger without getting the approval of other organizations. Any changes to the channel structure need to be approved by a set of organizations that can satisfy the channel policies.

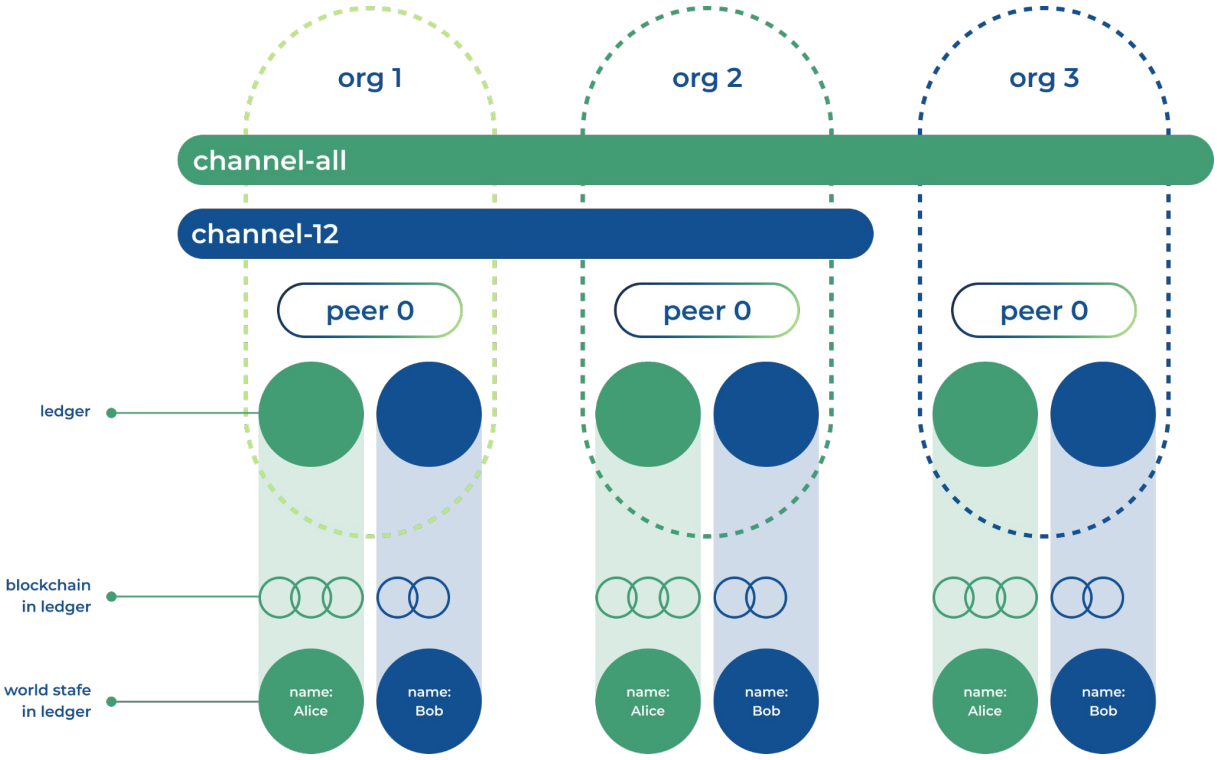

Figure 7: BusyXChain channels and organizations

A channel is defined by members (organizations), anchor peers per member, shared ledger, chaincode application(s) and the ordering service node(s). Each transaction on the network is executed on a channel, where each party must be authenticated and authorized to transact on that channel. Each peer that joins a channel has its own identity given by a membership services provider (MSP), which authenticates each peer to its channel peers and services.

The channels can mainly help with privacy, as clients connecting to one channel are unaware of the existence of other channels. Channels can be:

- private (hidden to all),
- public (open to all),
- hybrid (open to specific channels),
- or consortium (open to multiple organizations in the channel).

BusyXChain allows all types.

As it is possible to run more chaincodes in one channel, it doesn't matter if the chaincode (smart contract) is an asset, NFT, GameFi token, DEX, blockchain protocol, supply chain or just an application running on the blockchain. BusyXChain can easily connect each of them using channels without developing other complicated technologies (like bridges, cross-chains, etc.). On top of that, it allows the connection of different types of environments and tokens, for example, DeFi and NFT projects, GameFi and Metaverse, Company and its NTFs, etc.

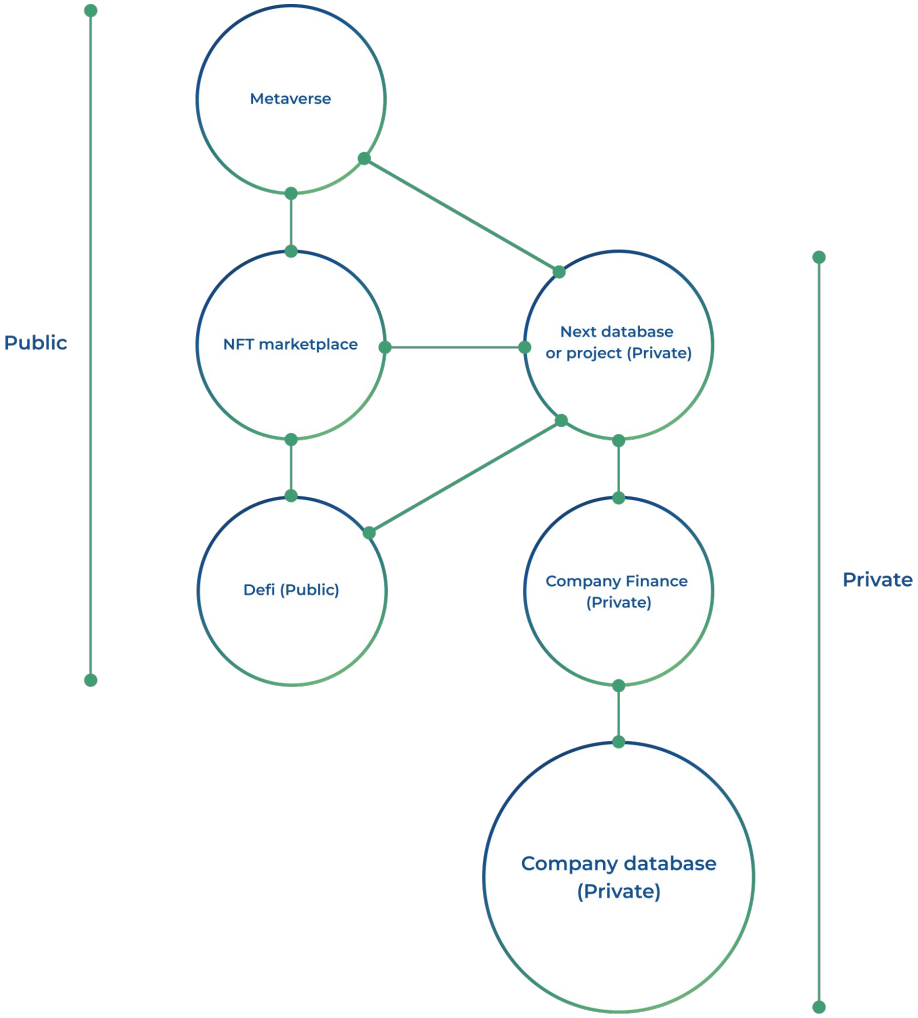

Figure 8: BusyXChain - Solution for Public and Private Companies

3.3.1. Privacy and interconnectivity

The channels can be seen as “independent” chains that add privacy. Transactions are done on a specific channel. However, Hyperledger Fabric is capable of more than just having multiple channels on one peer. Within the chaincode, it is possible to call other chaincodes. And, if the peer is part of the channel and the chaincode is installed on the peer, it is possible to call chaincodes from other channels. But no transaction is added to the ledger on the other channel. So it is possible to make only queries (reads) in the other chaincode.

Example (Figure 9)

The asset transfer private data example contains a collections_config.json file that defines three private data collection definitions: assetCollection, Org1MSPPrivateCollection, and Org2MSPPrivateCollection.

The policy in the assetCollection definition specifies that both Org1 and Org2 can store the collection on their peers. The memberOnlyRead and memberOnlyWrite parameters are used to specify that only Org1 and Org2 clients can read and write to this collection.

The Org1MSPPrivateCollection collection allows only peers of Org1 to have the private data in their private database, while the Org2MSPPrivateCollection collection can only be stored by the peers of Org2. The endorsementPolicy parameter is used to create a collection specific endorsement policy. Each update to Org1MSPPrivateCollection or Org2MSPPrivateCollection needs to be endorsed by the organization that stores the collection on their peers.

```
// collections_config.json

[
    {
    "name": "assetCollection",
    "policy": "OR('Org1MSP.member', 'Org2MSP.member')",
    "requiredPeerCount": 1,
    "maxPeerCount": 1,
    "blockToLive":1000000,
    "memberOnlyRead": true,
    "memberOnlyWrite": true
    },
    {
    "name": "Org1MSPPrivateCollection",
    "policy": "OR('Org1MSP.member')",
    "requiredPeerCount": 0,
    "maxPeerCount": 1,
    "blockToLive":3,
    "memberOnlyRead": true,
    "memberOnlyWrite": false,
    "endorsementPolicy": {
        "signaturePolicy": "OR('Org1MSP.member')"
    }
    },
    {
    "name": "Org2MSPPrivateCollection",
    "policy": "OR('Org2MSP.member')",
    "requiredPeerCount": 0,
    "maxPeerCount": 1,
    "blockToLive":3,
    "memberOnlyRead": true,
    "memberOnlyWrite": false,
    "endorsementPolicy": {
        "signaturePolicy": "OR('Org2MSP.member')"
    }
    }
]
```

Figure 9: Example collections_config.json

Example 2 (Figure #)

Specifically, access to the private data will be restricted as follows:

objectType, color, size, and owner are stored in assetCollection and hence will be visible to members of the channel per the definition in the collection policy (Org1 and Org2).

AppraisedValue of an asset is stored in collection Org1MSPPrivateCollection or Org2MSPPrivateCollection, depending on the owner of the asset. The value is only accessible to the users who belong to the organization that can store the collection.

```
// Peers in Org1 and Org2 will have this private data in a side database
type Asset struct {
       Type  string `json:"objectType"` //Type is used to distinguish the various types of objects
       ID    string `json:"assetID"`
       Color string `json:"color"`
       Size  int    `json:"size"`
       Owner string `json:"owner"`
}

// AssetPrivateDetails describes details that are private to owners

// Only peers in Org1 will have this private data in a side database
type AssetPrivateDetails struct {
       ID            string `json:"assetID"`
       AppraisedValue int    `json:"appraisedValue"`
}

// Only peers in Org2 will have this private data in a side database
type AssetPrivateDetails struct {
       ID            string `json:"assetID"`
       AppraisedValue int    `json:"appraisedValue"`
}
```

Figure 10: Example organization restriction

3.3.2. Permissions and roles (policies)

Endorsement policy specification

An endorsement policy is a condition on what endorses a transaction. BusyXChain blockchain's Peers have a pre-specified set of endorsement policies referenced by a deploy transaction that installs a specific chaincode. Endorsement policies can be parametrized, and these parameters can be specified by a deploy transaction.

To guarantee blockchain and security properties, the endorsement policies should be a set of proven policies with limited functions to ensure bounded execution time (termination), determinism, performance and security guarantees.

Dynamic addition of endorsement policies (e.g., by deploy transaction on chaincode deploy time) is very sensitive regarding bounded policy evaluation time (termination), determinism, performance and security guarantees. Therefore, the dynamic addition of endorsement policies is not allowed but can be supported in future.

Transaction evaluation against endorsement policy

A transaction is declared valid only if it has been endorsed according to the policy. An invoked transaction for a chaincode will first have to obtain an endorsement that satisfies the chaincode's policy, or it will not be committed. This takes place through the interaction between the submitting client and Endorsing Peers.

Formally the endorsement policy is a predicate on the endorsement and potentially further state that evaluates to TRUE or FALSE. For deploy transactions, the endorsement is obtained according to a system-wide policy (for example, from the system chaincode).

An endorsement policy predicate refers to certain variables. Potentially it may refer to:

- keys or identities relating to the chaincode (found in the metadata of the chaincode), for example, a set of endorsers;
- further metadata of the chaincode;
- elements of the endorsement and endorsement.transaction-proposal;
- and potentially more.

The above list is ordered by increasing expressiveness and complexity; that is, it will be relatively simple to support policies that only refer to keys and identities of nodes.

The evaluation of an endorsement policy predicate must be deterministic. Every peer shall evaluate an endorsement locally so that a peer does not need to interact with other peers. Yet, all correct peers evaluate the endorsement policy in the same way.

Example of endorsement policies

The predicate may contain logical expressions and evaluates to TRUE or FALSE. Typically the condition will use digital signatures on the transaction invocation issued by endorsing peers for the chaincode.

Suppose the chaincode specifies the endorser set E = {Alice, Bob, Charlie, Dave, Eve, Frank, George}.

Some example policies:

- A valid signature on the same transaction-proposal from all members of E.
- A valid signature from any single member of E.
- Valid signatures on the same transaction-proposal from endorsing peers according to the condition (Alice OR Bob) AND (any two of Charlie, Dave, Eve, Frank, George).
- Valid signatures on the same transaction-proposal by any 5 out of the 7 endorsers. (More generally, for chaincode with n > 3f endorsers valid signatures by any 2f+1 out of the n endorsers, or by any group of more than (n+f)/2 endorsers.)

Suppose there is an assignment of “stake” or “weights” to the endorsers, like {Alice=49, Bob=15, Charlie=15, Dave=10, Eve=7, Frank=3, George=1}, where the total stake is 100: The policy requires valid signatures from a set that has a

majority of the stake (i.e., a group with combined stake strictly more than 50), such as {Alice, X} with any X different from George, or {everyone together except Alice}. And so on.

The assignment of stake in the previous example condition could be static (fixed in the metadata of the chaincode) or dynamic (e.g., dependent on the state of the chaincode and be modified during the execution).

Valid signatures from (Alice OR Bob) on tran-proposal1 and valid signatures from (any two of Charlie, Dave, Eve, Frank, or George) on tran-proposal2, where tran-proposal1 and tran-proposal2 differ only in their endorsing peers and state updates.

How useful these policies are will depend on the application, on the desired resilience of the solution against failures or misbehavior of endorsers, and on various other properties.

3.4. Chaincode (Smart contracts)

A chaincode typically handles business logic agreed to by network members so that it may be considered a “smart contract”. Ledger updates created by a chaincode are scoped exclusively to that chaincode and can’t be accessed directly by another chaincode. However, given the appropriate permission within the same network, a chaincode may invoke another chaincode to access its state. Thanks to this option, everyone can automate and digitize their project in the BusyXChain network.

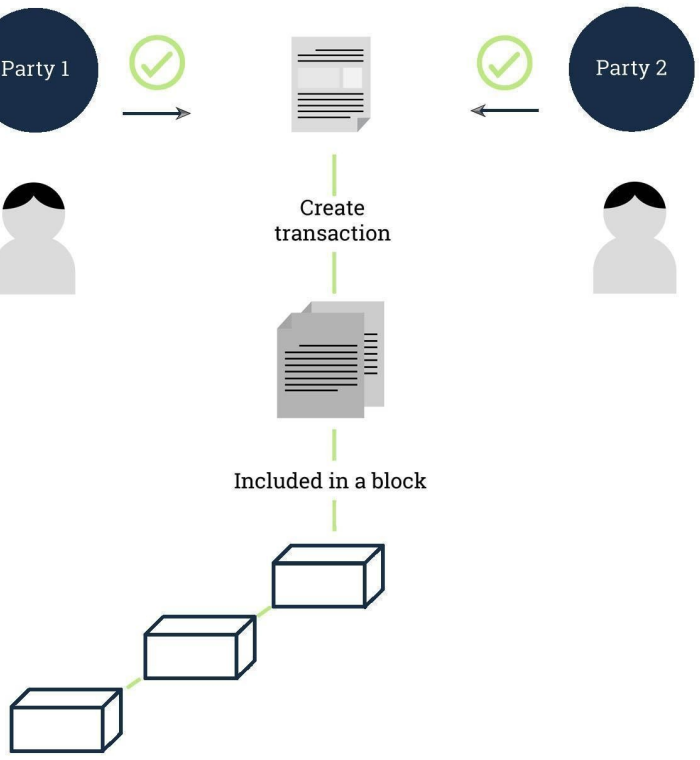

Figure 11: Smart contact flow diagram

Three key points apply to smart contracts:

- many smart contracts run concurrently in the network,
- they may be deployed dynamically (in many cases by anyone),
- application code should be treated as untrusted, potentially even malicious.

Most existing smart-contract capable blockchain platforms follow an order-execute architecture in which the consensus protocol:

- validates and orders transactions, then propagate them to all peer nodes,
- each peer then executes the transactions sequentially.

The order-execute architecture can be found virtually in all existing blockchain systems – from public/permissionless platforms such as Ethereum (before with PoW-based consensus) to permissioned platforms such as Tendermint, Chain, and Quorum.

Smart contracts executing in a blockchain with the order-execute architecture must be deterministic; otherwise, they might never reach a consensus. To address the non-determinism issue, many platforms must be written in a non-standard or domain-specific language (such as Solidity) to eliminate non-deterministic operations. This hinders widespread adoption because it requires developers to write smart contracts to learn a new language and may lead to programming errors.

On BusyXChain, developers can write chaincodes in classic programming languages like Go, Node.js, or Java that implement a prescribed interface.

BusyXChain uses architecture for transactions that we call execute-order-validate. It addresses the resiliency, flexibility, scalability, performance and confidentiality challenges faced by the order-execute model by separating the transaction flow into three steps:

- execute a transaction and check its correctness, thereby endorsing it,
- order transactions via a (pluggable) consensus protocol,
- validate transactions against an application-specific endorsement policy before committing them to the ledger.

An application-specific endorsement policy specifies which peer nodes, or how many of them, need to vouch for the correct execution of a given smart contract. Thus, each transaction needs only be executed (endorsed) by the subset of the peer nodes necessary to satisfy the transaction’s endorsement policy. This allows for parallel execution (called sharding), increasing the overall performance and

scale of the system. This first phase eliminates non-determinism, as it filters out inconsistent results before ordering.

Thanks to the features mentioned above, the overall complexity of calculations and the entire network is lower, so it is ideal as a platform for various uses.

3.5. $BXCH Coin

BusyXChain comes with its mainnet asset/coin called $BXCH. After the initial funding round and public sale, all investors and users will receive the mainnet $BXCH coin directly.

The coin will function as the main ecosystem currency. Users and developers will be paying for creating the chaincodes (smart contracts), regular transaction fees, and fees for creating custom tokens. All fees are burned as part of the deflationary mechanism. On the other hand, staking will mint $BXCH for the users holding/staking their coins.

3.6. X Tokens

BusyXChain supports three types of tokens. These tokens have only basic functions specified according to certain standards. More types of tokens will be added based on the demand. All these types of tokens will take advantage of the BusyXChain network.

3.6.1. X20

X20 token is programmed the same as ERC-20 Token Standard. Ideal for DEFI projects and stable coins. This type usually represents "classic" (crypto) assets.

3.6.2. NFT

NFT token is programmed the same as the ERC-1155 Multi Token Standard. This token type is known as original or unique, with a digital identifier that is recorded in a blockchain that cannot be copied, substituted, or subdivided. NFTs are becoming sought after as collectables. NFTs can be hard-coded to digitally store and protect important documents like degrees, medical history, licenses, arts, etc.

3.6.3. GAMEFI (Metaverse)

The GAMEFI token is programmed the same as the ERC-1155 Multi Token Standard, similar to the NFT token. It is used to create non-duplicable in-game items that enable gamers to verify the authentic owner of the collectable easily. Then these unique in-game items can be, for example, traded on marketplaces and auctions.

3.6.4. Custom token

Custom token (smart contract) can be written / programmed by the user to cover their special needs and then work/run on BusyXChain network. This token can be developed in multiple classic languages like go, java, and node.js; then can be deployed on the BusyXchain network. In the beginning via command line (cli / cmd) and API, in the next stage via the online SDK automated tool.

3.7. Use and Build Cases

In this section, we will show a few uses and build cases that can be easily built using BusyXChain. It has to be said that holding a token is not a must. A company does not need to create a custom token to create a channel and application.

The use cases are almost unlimited, these are only a few examples:

- Various projects (from small business to global corporations),
- public and private company networks,
- sharing data between two or multiple companies and projects.

A few build cases that the BusyXChain solution can be used, for example:

- DAO – Any DAO can run on BusyXChain. Using the option of setting endorsement policies (see section 3.2.2.), guarantees that everything must always be approved by a majority.
- DeFi – DEFI projects can easily use no bridge and cross-chain policy to become accessible to everyone
- E-commerce software – Any e-commerce model can be easily applied with advantages like connecting with an existing solution and business partners without major changes to the company's infrastructure.
- DEX – decentralized exchanges can also run on the BusyXChain protocol, where the connectivity of individual assets and low fees will be a big advantage.
- Supply chains – are global or regional webs of suppliers, manufacturers, and retailers of a particular product. BusyXChain networks can improve the transaction processes of the supply chain by increasing the clarity and traceability of transactions. On a BusyXChain network, enterprises having authentication to access the ledger can view the data of previous transactions. This fact increases accountability and reduces the risk of counterfeiting transactions. Real-time production and shipping updates can be updated to the ledger. Which can help track the product condition in a much faster, simpler, and more efficient way.
- Financial instruments (Trading and Asset Transfer) – Trading and asset transfer requires many organizations or members like importers, exporters, banks, and brokers. They work with one another. And even in

the era of digitalization, a lot of paperwork needs to be done in this sector. But using BusyXChain, they can transact and interact with each other in a paperless environment. BusyXChain can add the same layer of trust as a document signed by a trusted authority. This also increases the performance of the system. Another benefit of BusyXChain is that assets can be dematerialized on the blockchain network. Due to this, traders or stakeholders will be able to have direct access to their financial securities, and they can trade it anytime.

- Insurance – The insurance industry spends billions to avoid fraud or falsified claims. With the help of BusyXChain, Insurance Companies can refer to the transaction data that is stored inside the ledger. BusyXChain can also make claims processing faster using the chaincode and automate the payment. This process will also be helpful for multi-party subrogation claims processing. Where it can automate repayment from the liable party back to the insurance company. Verification of identity or KYC process will be easy using this private blockchain.

3.7.1. Example #1 – Private Company and its clients

Requirements and proposal

Let’s say we are building an application for a Financial Company. They came to us and shared that they handle many clients and need to exchange many paper documents before the interested parties are ready to sign any of them. Unfortunately, some records are lost, and sometimes there are changes someone was unaware of; in short, they have many problems everyone would like to avoid. On top of that, clients get mad, so we have to do something, or the company will start losing clients.

We proposed to build an application based on blockchain technology, so the company and their clients would be able to cooperate on documents, track all changes, and verify document integrity at any point in time. That way, they can avoid many misunderstandings.

After a surprisingly short amount of time, their biggest client agreed to participate. Let's name them Company A. All network pieces are in place, but it is necessary to focus on peer nodes. Peer nodes will have copies of the ledger where the application will store data, like an audit of changes made to the documents. The client needs to have peer node/s on their infrastructure (or hosted) to ensure that the data is genuinely valid. Then, the client must join peer node/s with the Financial Company's newly created application channel, where all relevant chaincodes are deployed.

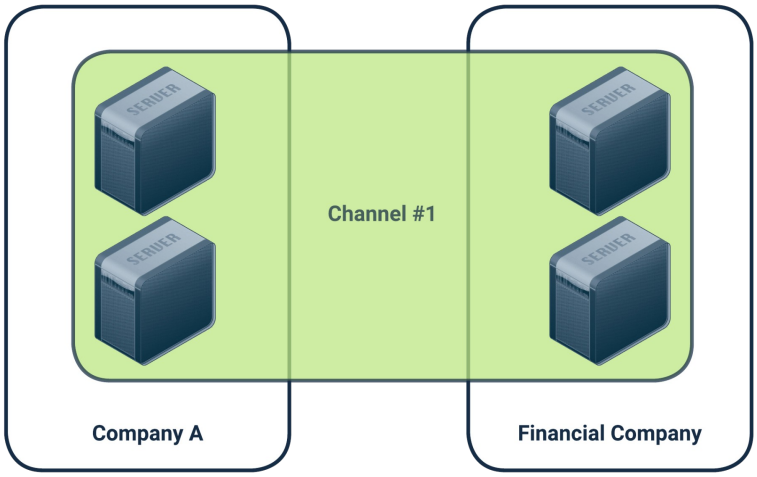

Figure 12: Use Case Example #1 – Company and one Client in one channel

The client admitted they are delighted with the way the application is working. Finally, there is no need to exchange paper documents, and they can easily keep track of everything.

More clients to be added?

The next step for the Financial Company is to implement this solution with the next client; let's call them Client B. The Financial Company has a working network with one application channel them and Company A use. They don't want to share that channel with Company B as they would be able to see everything from the ledger and maybe even write data to it too. Next is necessary to add peer nodes to the infrastructure owned by Company B and create a new channel that will join the Financial Company's and Company B's peers. In that way, they will have two ledgers, one for each client, but only peers that are members of the channel will have a copy of their ledger.

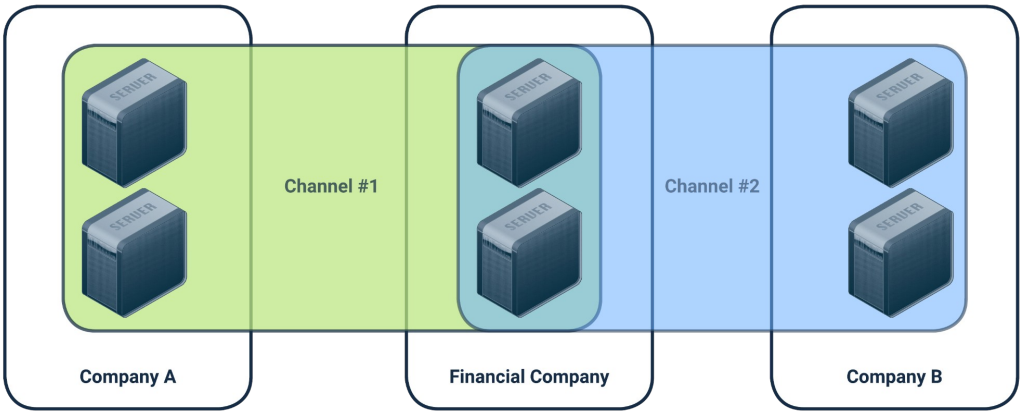

Figure 13: Use Case Example #1 – Company and two Clients in two channels

Every time the solution is implemented with other clients, they need to set up peer node/s on their side, then join a newly created channel, and the ledger is stored only on the participating peer node/s. Though in our example, we use a channel to communicate just between two organizations, adding more parties to a single channel is possible.

Scalability (sharding)

As word about the Financial Company's application spreads worldwide, more and more clients are interested in using it, and they also landed a few new clients.

Thankfully, each peer node/s can be a member in different channels. Given that at the beginning, the Financial Company had three peers and each of them was a member of two separate channels, now they can spawn more peer nodes and utilize them with new clients. That way, they can achieve "sharding". For the sake of the simplicity of the diagram, let's say the Financial Company has four peer nodes and three clients. We can configure the network so each channel will be placed on two of their peers.

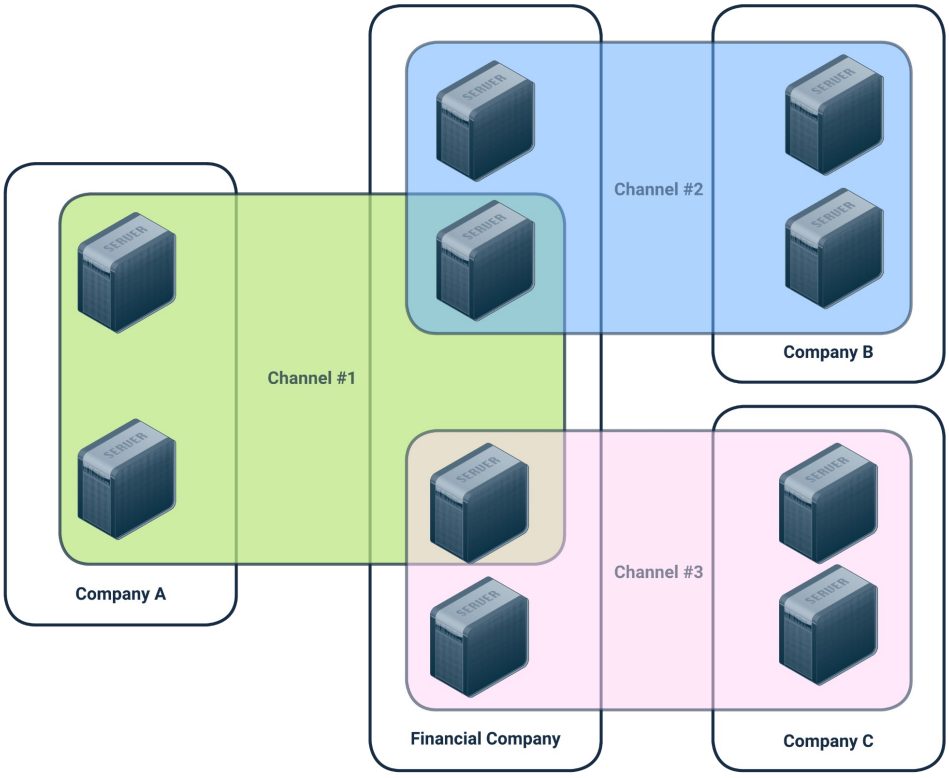

Figure 14: Use Case Example #1 – Company and three Clients in three channels

Summary

As you can see, a channel is an essential building block of their network. Without it, they cannot build one. It also securely separates data, so only peers of a particular channel will store the data, and only clients with proper permissions

can read and write to the channel's ledger. The last thing to highlight is that they can freely configure their peers so that each can know about different channels. In that way, the network can be more fail-proof or can achieve sharding.

3.7.2. Example #2 – Supply chain companies

The model below shows a supplier, manufacturer, warehouse, hospital, and admin. Each entity is designed to acquire a copy of the central ledger. The blockchain's stored data cannot tamper with; consequently, leaking confidential data will be impossible.

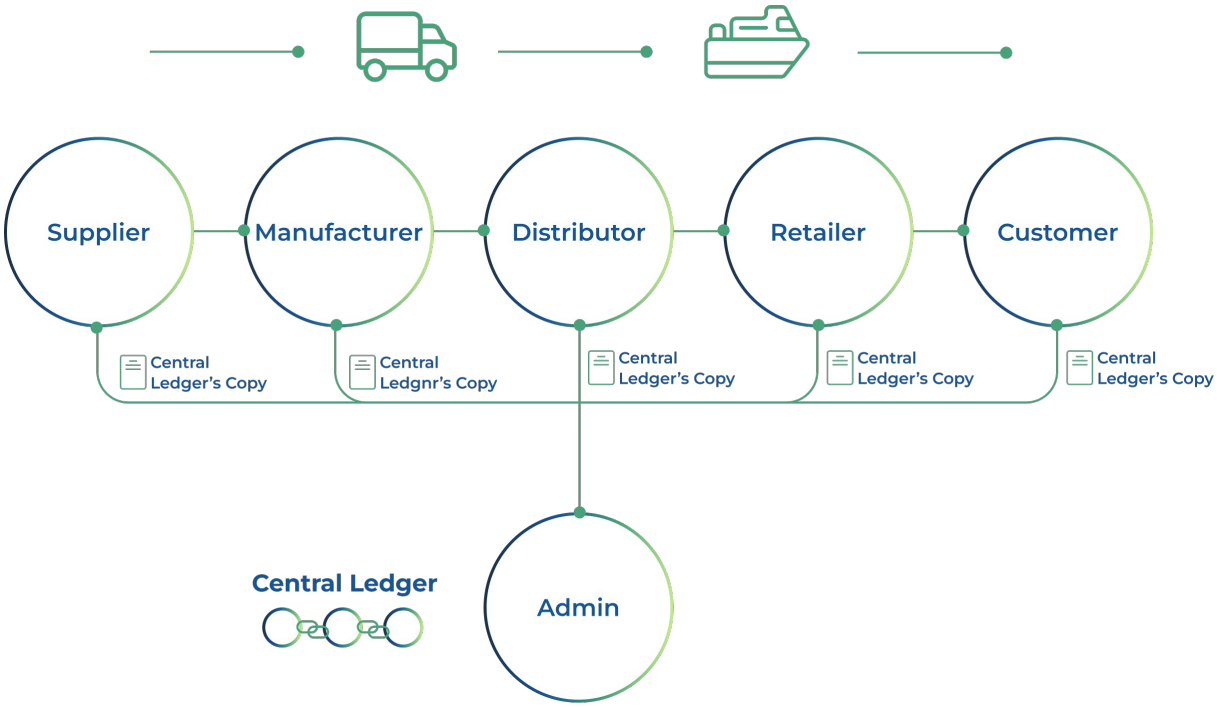

Figure 15: Use Case Example #2 – Supply Chain Company

In this scenario, five entities need to work and share data. One of them (The Distributor) has an admin role; the others have just specific roles. Each role can write only a set of predefined data (for example, supplier - raw material, variety, and quality), but everyone can read this. That means the admin creates base data, and other members can only check and update this data. It is based on read/write rules in endorsement policies.

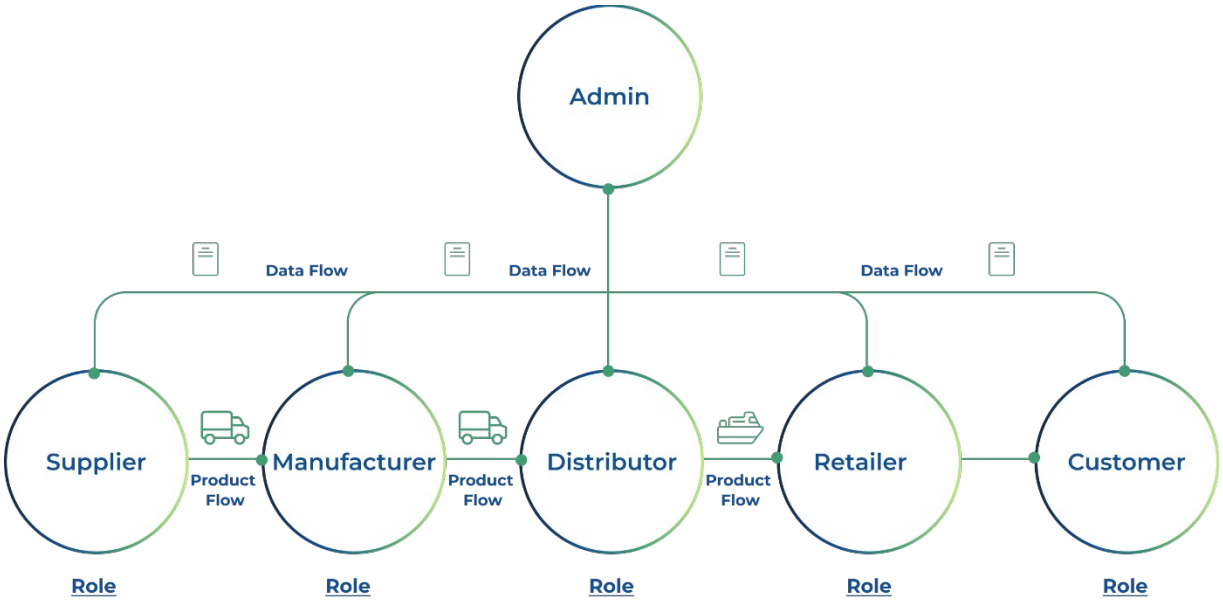

Figure 16: Use Case Example #2 – Supply Chain Company – Roles

In this case Admin have a read/write role the others have only read role and can invoke just one specific function and input data in prepared schema.

4. Products

Busy Technology already has some products in development and others on a plan for the BusyXChain.

4.1. BusyXChain blockchain

BusyXChain blockchain already exists. Busy Technology Company has already invested financial resources in the development and deployment of the blockchain which runs on Kubernetes. Currently, there are two branches – production and development.

4.2. Explorer

The basis for every blockchain project is a data explorer. Explorer is usually a web application that shows all the public data from the blockchain to the users in a user-friendly way. BusyXChain already has its explorer which is connected via API to the BusyXChain network.

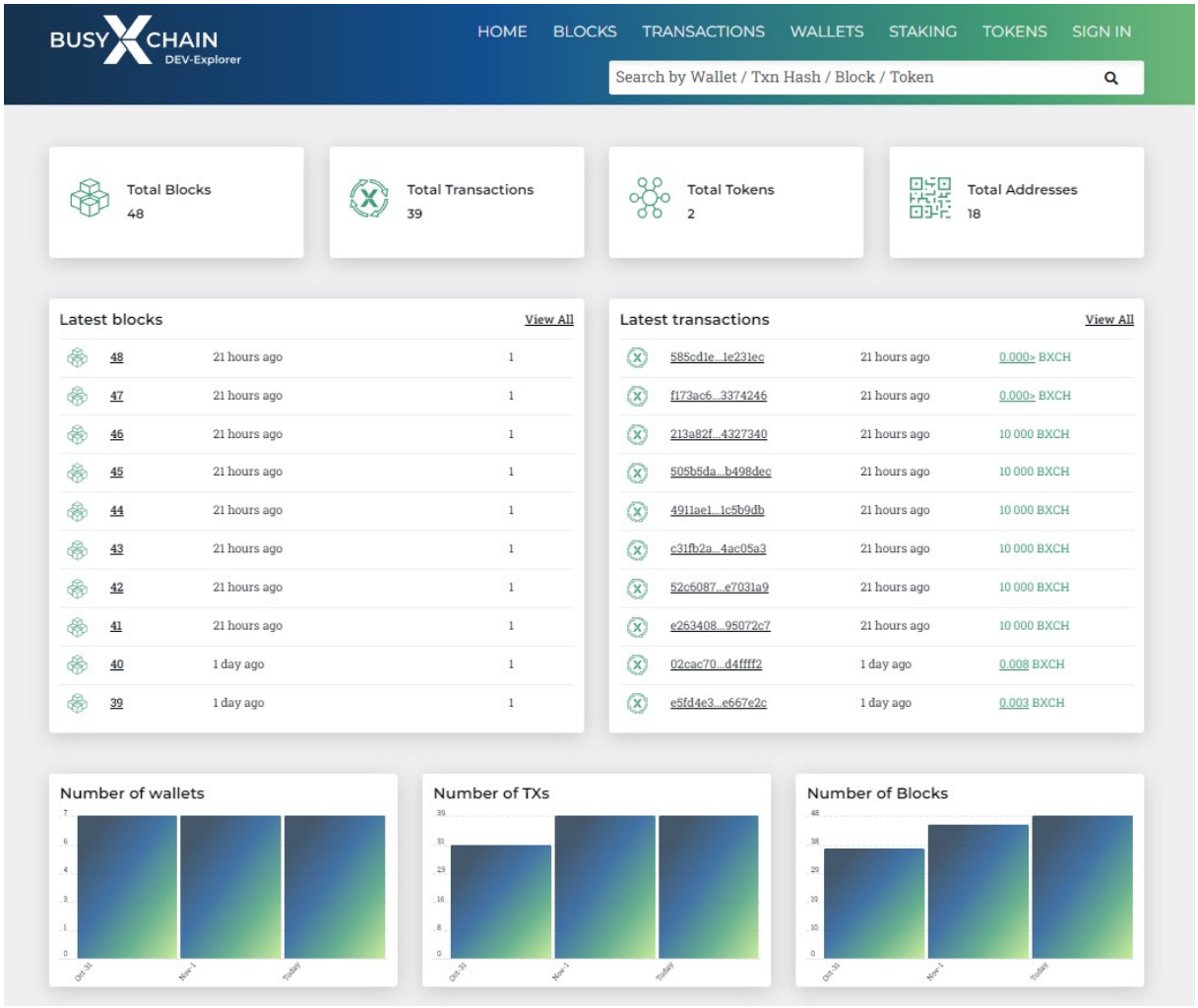

Figure 17: Products – Explorer – Dashboard

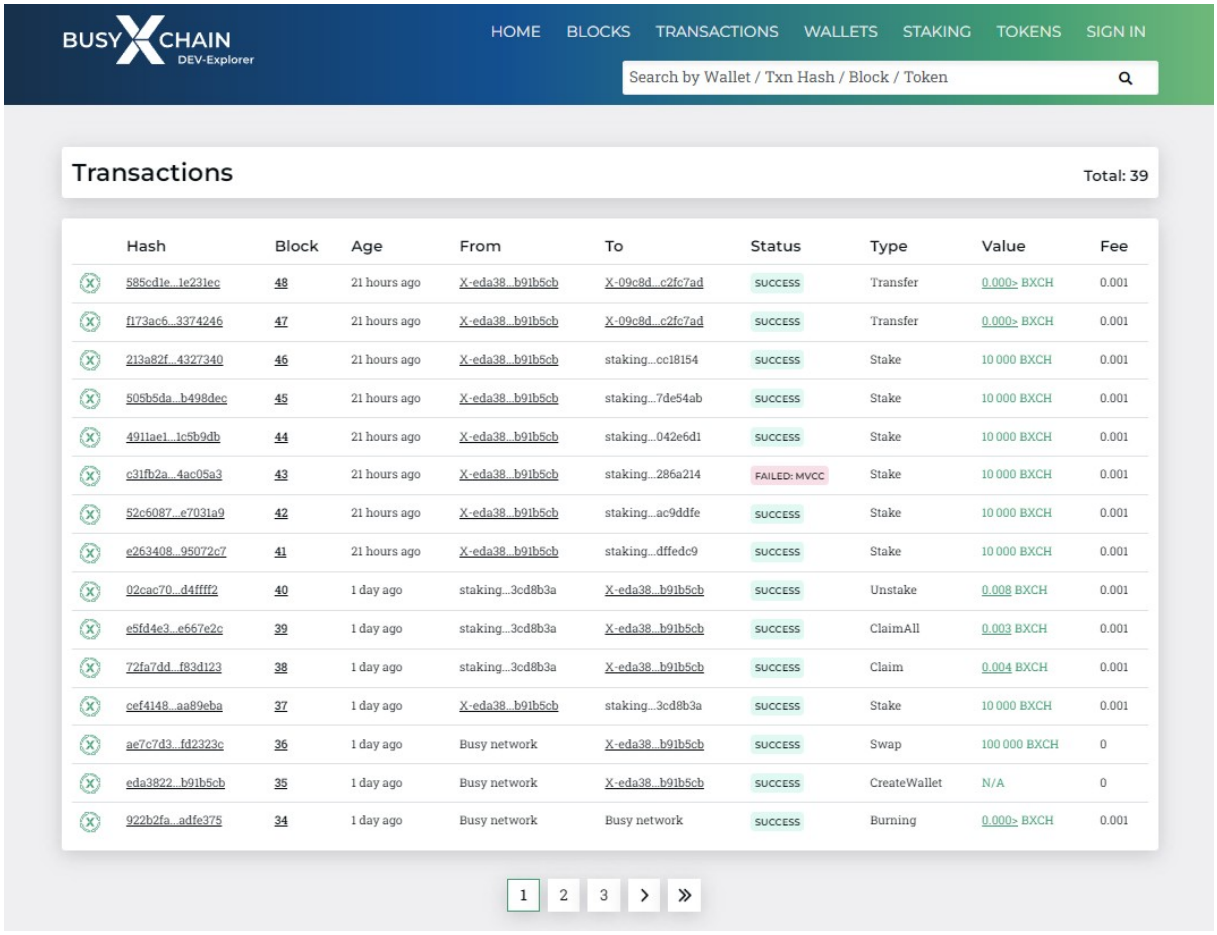

Figure 18: Products – Explorer – Transactions

4.3. Desktop Wallet with Token Creator

Another tool for our users is the BusyXChain desktop wallet. It allows users to manage their BusyXChain assets, storing locally encrypted private keys and data. The Desktop Wallet has already been released on the testnet network. Also, two rounds of Community Testing have been completed. It is available for major operating systems: Windows, macOS, and Linux. The biggest advantage against other wallets is that users can quickly create their X20, NFT, or GAME custom tokens in the inbuilt Token Creator.

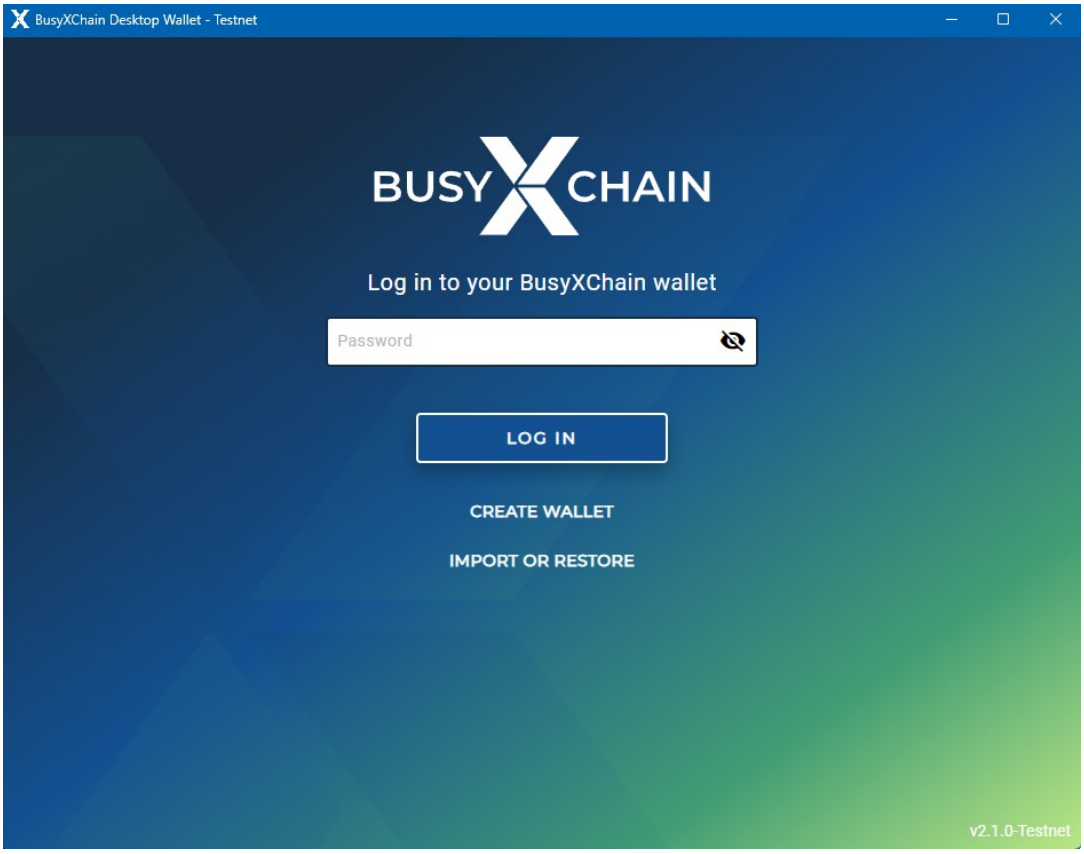

Figure 19: Products – Desktop Wallet

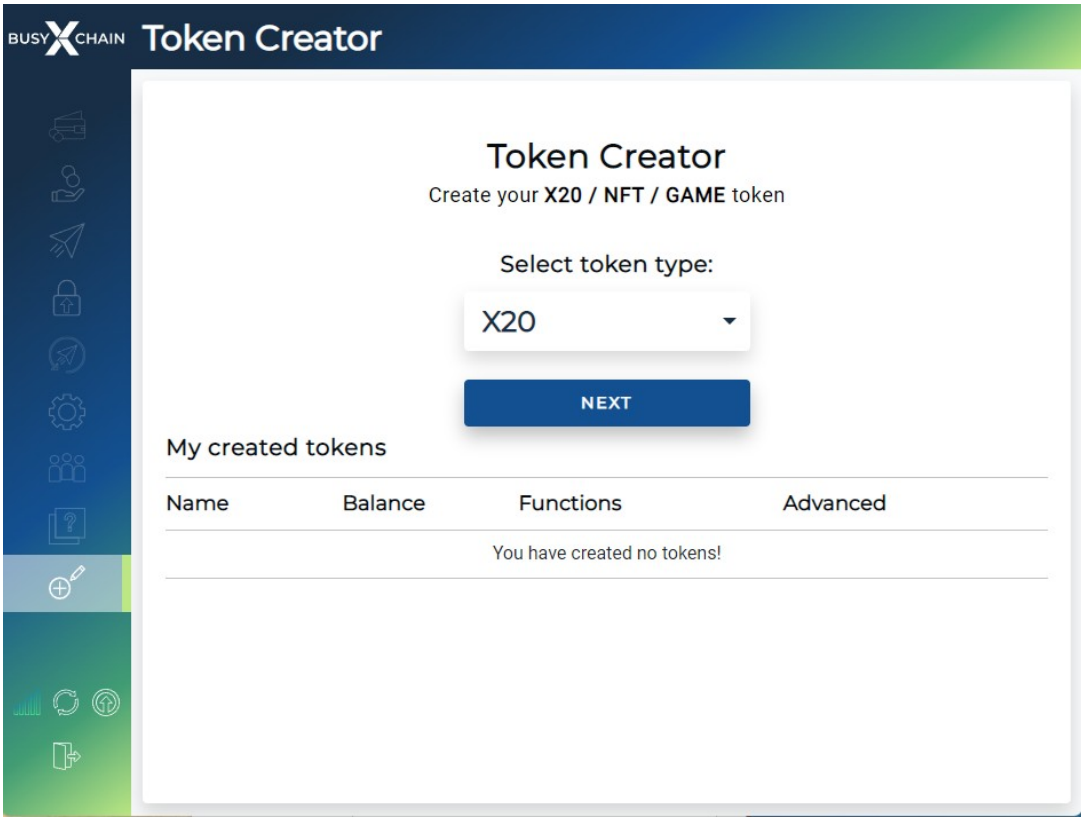

Figure 20: Products – Desktop Wallet – Token Creator – Dashboard

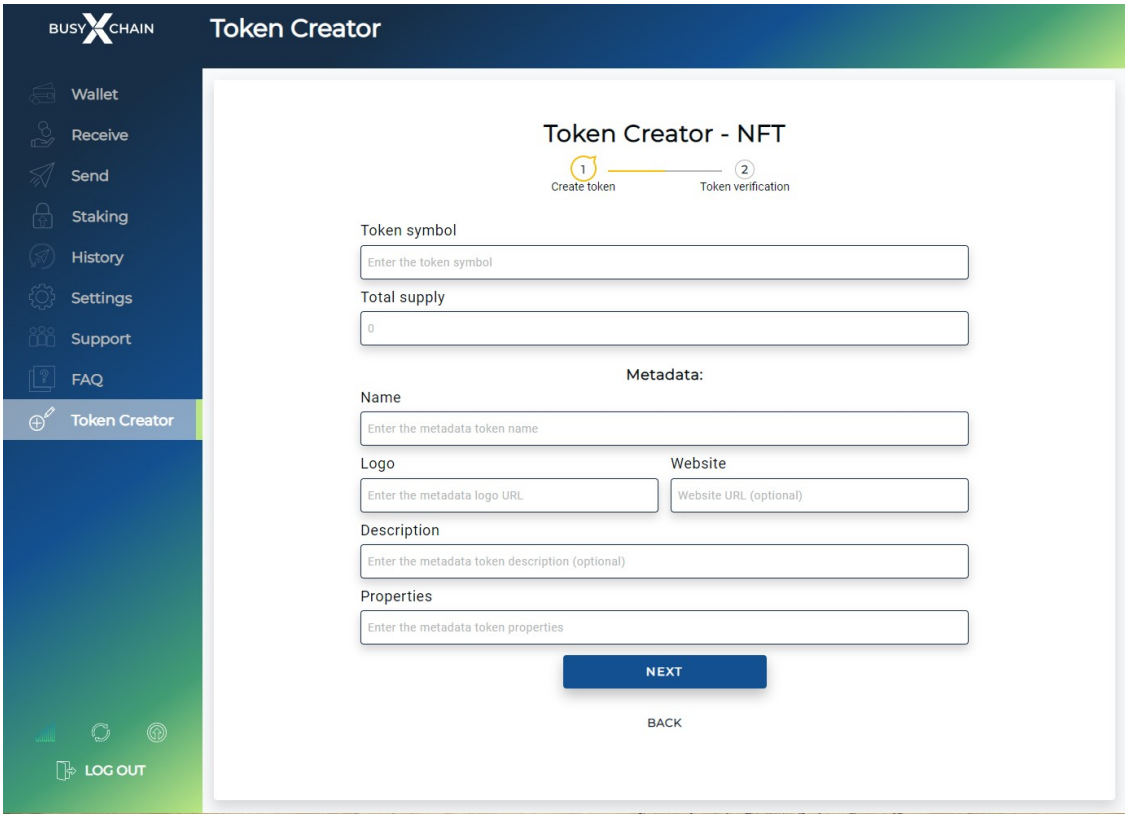

Figure 21: Products – Desktop Wallet – Token Creator – Create NFT

4.4. Mobile Crypto Wallet

In addition to the desktop wallet, Busy Technology also wants to develop a crypto mobile application that will behave the same as the BusyXChain Desktop Wallet. Furthermore, in the future the mobile application will work as a transaction confirmation processor (like two-factor authentication). Transactions performed on external services and projects developed on BusyXChain will need to be approved on the mobile application.

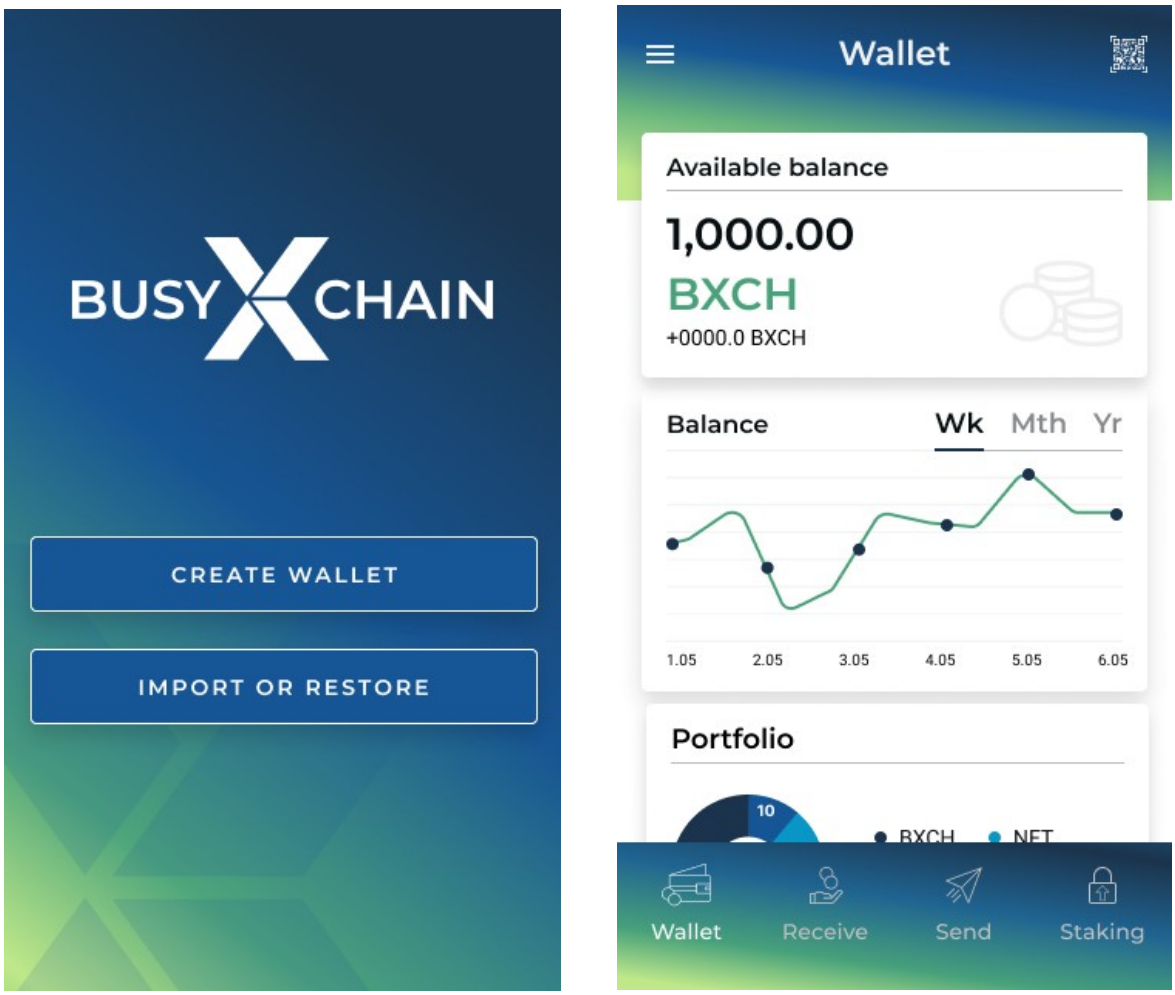

Figure 22: Products – Mobile Crypto Wallet

4.5. Online Automatic SDK Deployment tools

BusyXChain blockchain is available to everyone, to make it user-friendly we are planning to develop an online automatic SDK deployment tool for smart contracts that will make it easier also for regular users to create channels and applications within the channels.

4.6. Token economy - coming soon

1. Conclusion

This white paper describes the principle and functionality of the BusyXChain network.

The BusyXChain is a modular network with the ability to join and connect with existing infrastructure or create a bespoke solution. The core focus is on smart contracts and token creation with utility in a broad spectrum, including DeFi, NFT, and GameFi and Metaverse. BusyXChain connects anyone with anyone and everything. BusyXChain offers nearly limitless ways to connect projects, different layers, and private and public chains, all in the layer-1 solution.

Any project, company, DAO, institution, or government can build on BusyXChain or join with their existing infrastructure (for example, ERP) and move the whole business or just a particular process to the blockchain. The significant advantage is that multiple organizations can co-exist in one ecosystem. Form a consortium and participate in network management and data sharing according to set rules. Organizations can set their own rules and visibility with others easily and securely. With policies, the project, company, or government can be completely invisible, fully visible or a combination of both based on the permissions.

BusyXChain eliminates the need for complicated, risky bridges or cross-chain integration mechanisms by using a virtual layer above our core blockchain. The virtual layer creates the interconnectivity between various projects and channels. All of which can be customized by users based on their own unique requirements.

2. Figures