# êvrynet

An Intelligent Financial Services Platform

# ABSTRACT

EvryNet is an intelligent financial service platform that provides open-source micro-banking services to the unbanked and underbanked populations. EvryNet will create automated escrow services using an interoperable smart contract platform that can facilitate the creation and execution of micro-banking services to anyone at a competitive price. By providing automated financial smart contracts, EvryNet provides accountability, auditability, and irrevocability to financial contracts running on its blockchain-based platform. EvryNet will enable trustless financial transactions among unknown parties.

With the EvryNet intelligent financial service platform, users and institutions can easily compose or customize financial smart contracts using EvryNet contract templates through the EvryNet Financial Service Portal. The composed smart contracts will be processed by the EvryNet Smart Contract Execution Engine running on EvryNet nodes. The execution result of each EvryNet node will be incorporated in the EvryNet Reputation Network, reinforcing the reputation of high-performing and honest nodes. The EvryNet platform enables automation of financial contract execution, resulting in better efficiency and affordability without compromising flexibility to tailor to individual or institutional needs.

# TABLE OF CONTENTS

# 1. INTRODUCTION

Financial automation is essential to enable the creation of micro-banking services and markets that are customized and responsive to the needs of the underbanked. Here, underbanked is defined as low-income households and those lacking savings insurance,credit instruments and also investment portfolios for retirement savings;village level entities and small to medium-sized enterprises (SMEs) needing inventory and invoice financing . The EvryNet platform facilitates applications that aim to make access and execution of financial services efficient and economical, where lack of trust is a barrier.

Banks have legacy infrastructures, yet many of them have failed to deliver affordable banking solutions for the underbanked. According to a study by the Asian Development Bank on financial access and digital solutions, the needs met by formal financial service providers in Indonesia, Philippines, Cambodia, and Myanmar range from 11-75% for payments, 16-74% for savings, 48-72% for credit, and 1-4% for insurance. Specifically, 55% to 90% of all payment transactions in Southeast Asia are conducted through physical cash payments. In other research, Alvarez, Pawasutipaisit, and Townsend calculate in “Cash Management in Village Economies” that the costs of cash mismanagement lie between 2% to 9.5% of monthly consumption, thus quite large. In sum, more than 70% of people living in Southeast Asia have no access to basic banking products. Peer-to-peer lending services have emerged, but these are still limited as underlying custodian banking services are still lacking. Both services and exchange markets need to be improved.

There is a way forward. The region’s 655 million population is highly connected through digital channels, and the internet user base is estimated at 400 million in 2020; 90% are connected to the internet via smartphones. This digital trend, coupled with blockchain technology, creates an opportunity for improving financial inclusion for Southeast Asia’s underbanked.

EvryNet’s intelligent financial service platform will enable low-cost micro-banking services, e.g., micro-Letter of Guarantee (LG), micro-Bank Guarantee (BG), micro-trade finance products, accessible investment portfolios, and exchange markets, which are essential to the growth of the region’s digital economy. The EvryNet platform is a distributed financial smart contract platform that allows people to leverage smart contract technology [1] to create and execute financial contracts in a trustless and frictionless manner. The platform provides simplified building blocks for smart contract creation, multi agent interactions, and exchange through EvryNet templates and SDKs/APIs. With EvryNet SDKs/APIs, EvryNet service providers can create customized financial smart contracts for micro-banking services.

EvryNet is built on the concept of peer-to-peer distributed network. Anyone or any organization can offer EvryNet nodes to join EvryNet. EvryNet nodes are responsible for smart contract execution and storage

in exchange of service fees. The more EvryNet nodes join the network, the more computing power and storage capacity there will be.

The EvryNet platform will create a smart contract economy where everyone is incentivized to contribute additional computing power and storage capacity to support more users. As a result, the EvryNet service fees will become more affordable because of the economy of scale.

EvryNet nodes are not required to trust one another in order to join the network. The EvryNet network embraces a reputation system, which is essential to guide the node selection process for contract execution. Additionally, EvryNet's reputation system can augment the credibility of individual users or EvryNet nodes. In the reputation system, all user accounts and EvryNet nodes in the EvryNet network are dynamically updated with reputation metrics based on behaviors, reliability, and contributions to the network. The evaluation of the contract execution outcome is also considered in the reputation calculation. EvryNet can leverage eKYC service [2], such as N1N0, to provide the chain of trust for each user account or EvryNet node, serving as a reputation bootstrap.

EvryNet smart contracts can optionally check for compliances or necessary regulations. For instance, the EvryNet platform can allow the relevant organizations to certify by digitally signing the contract or even executing the compliance-check code to ensure regulatory compliance. EvryNet also offers encryption to store smart contracts privately and leverage the concept of private channels [3] when all relevant parties enter the smart contract execution.

Many real-world smart contracts may need external inputs to complete the conditional transactions [4][5] — like a confirmation of a shipment for instance. The EvryNet platform allows event hooking in smart contracts to seamlessly receive relevant events from external entities. The platform provides trackability for transparency throughout the network. For example, EvryNet can track which EvryNet nodes or services, and which user accounts participate in a smart contract at any stage of execution. This enables EvryNet to support peer-to-peer transactions, such as: transfers, custodian services, and auction services. In a property lending scenario, EvryNet will validate a property title transferred to the escrow account and ensure the lien is removed after the loan is paid back in the future. Such titles can be transferred in primary and secondary markets. In a waterfall payment scenario, EvryNet eliminates the need to trust that the payroll provider will pay and that the payment bank will be solvent. Similarly, employees can borrow against future payroll receipts, and these loans can be tokenized and traded in secondary markets. Aside from the trustless environment that EvryNet offers, the expressivity of its smart contracts will boost financial service innovation as developers or service providers will not be required to have specific knowledge to initiate and execute financial contracts.

# 2. ECONOMIC BACKGROUND

## 2.1 Micro-Finance

To summarize financial access and informal markets, here, we will briefly cover the Townsend Thai Project, an in-depth study specifically focused on the topic. In the study, we learn how the overall sharing of risk to mitigate impacts of shocks onto consumption and production is good, but not perfect. Idiosyncratic household and business specific shocks are large, occur frequently, and constitute the bulk of the risk in a village economy. Notably, these idiosyncratic shocks are mostly, though not entirely, pooled away [1].

The mechanisms of shocks are when local village or tambon (county) level "money markets" wherein households borrow to repay other pre-existing loans, then borrow to lend, accruing new loans. These behaviors are shown to be one source of consumption smoothing — an economic concept of human behavior used to express the desire to have a stable path of consumption, in which consumption indicates expenditures [2].

One problem is delayed repayment, where repayment on loans goes back through the credit chain, causing delays for those who have borrowed to lend. On the other hand, early repayment could also go back through the credit chain, even more so when kinship and transaction-based links are involved [3].

Repayment through kinship will be referred to as risk sharing, which is largely dependent on family/kin and proximity of ancestry trees. Aggregate village-level or tambon-level shocks by definition are not diversifiable at the village level and end up in consumption. Like sophisticated financial markets, households with self-financed businesses or projects with self-financed or borrowed assets have return rates that reflect risk premia and influence of risk onto production, hence impact of the aggregate shocks. By the same standard, aggregate shocks across villages could be, but are not much pooled [4].

Another factor to consider are geographic limitations. In 2002, the government of Thailand put a Million Baht Fund in every village, creating a quasi-formal village-level institution to be run by a local committee.

---

[1] Chiappori, Samphantharak, Schulhofer-Wohl and Townsend, 2014

[2] Sripakdeevong and Townsend, 2018

[3] Kinnan, Samphantharak, Townsend, and Vera-Cossio, 2018

[4] Samphantharak and Townsend, 2018

The pre-existing baselines and post-intervention panel data allowed insights to how influx of capital impacted the villages and households and levels of consumption increase, among other variables [5].

The crux of the finding was that investment and profits of those already in non-agricultural businesses increased if they had measured high rates of return in the baseline [6]. The informal sector acted as a catalyst, playing an augmenting, complementary role. For the relatively poor, the lowest wealth quartile, the best-fitting financial regime shifted from a buffer-stock-saving regime to a costly-state-verification regime. Verification costs are lower when there are kinship households in the village [7].

However, there is still room for improvement.

Kinship ties can be a mixed blessing when considering the following hypotheticals: what if there are no kin(s) nearby? What if village fund committees do not allocate funds efficiently? There are also no statistical connections to rate of return, and allocation is based more on family and political connections [8]. Some of the village fund loan money made its way indirectly to other non-direct-borrowing households, again through kin connections and network credit chains. But historical data show village funds did not reach all of those who should have been ex-ante optimal targets, or simply put, there is risk of mismanaged fund allocations with little to no accountability, and a bias towards kinship. More generally households running SMEs remain vulnerable to some idiosyncratic risk which creates distortions in long term life cycle savings and investment behavior.

In short, what is needed are better allocation mechanisms within villages among non-kin, not relying on trust, and better markets across villages. The village example is both real and a metaphor. Households and SMEs in urban areas including Bangkok suffer from distortions.

## 2.2 Contracts

Here we review the basics of a contract; for example, but not limited to: stipulations, conditionality, and outcomes to the parties. The relevance of contracts comes to life and is super charged with the possibilities created by the new distributed ledger technology, commonly termed “smart contracts”. This section summarizes and outlines the various meanings of the word trust, as with trust in third parties, needed or not.

[5] Kaboski and Townsend, 2011-2012

[6] Banerjee, Breza, Townsend, Vera-Cossio, 2018

[7] Ru and Townsend, 2018

[8] Vera-Cossio, 2018

The simplest form of contracts between two parties is the purchase and sale of a commodity or asset. If transacted as a spot market exchange, the language of contracts could be omitted, but on the blockchain, there is a need to define the contract as an agreement of the buyer to surrender cash and the seller to surrender the commodity (the buyer is debited, and the seller is credited).

When there are lags between the time of the agreement to trade and the eventual payment, issues could arise. If there is no collateral, the parties trust each other to carry out their part, commonly referred to as a full commitment contract, where promises are made and honored, assumed in the contract literature, hence, one notion of trust. An alternative method is for a third party to stand between the traders making trade possible among strangers. Banks and dealers can do this, for example, but as noted this kind of legacy infrastructure has limited outreach so far and typically display markups that advantage profits and not client welfare.

With borrowing and lending, trust requirements are more apparent, as the lender either trusts the borrower to repay or there is collateral backing the loan e.g., in escrow. In the latter case, a transaction among strangers is possible when implemented with a conditional ‘if-then’ statement to define what happens with or without default. This includes borrowing liquidity as an SME for investment or smoothing or borrowing using one asset as collateral to acquire another.

When there is private information known by only one of the parties, then others may trust the informed party to send a message and reveal it, but typically in the contract mechanism design literature, the contract has to be designed in such a way to induce the informed party to tell the truth. In that case, trust, per se, is not needed. Where there is an unobserved effort or other unobserved actions, actions are recommended. Either a party is trusted to take appropriate actions or must be given an incentive to do so. This is the classic moral hazard problem.

A third example is when output from a project is privately observed. Verification of a project output by a lender or insurer is possible but costly. Over a range of outputs, repayment of a loan is constant, resembling debt and actual outcomes need not be known. For low outputs, claims are verified at a cost, as if validating financial statements and state verification is costly.

Given these examples, the word trust could be confounded with information and incentive issues, but on the other hand, from contract theory, we know these are all in fact separate issues. In particular, incentives are separated from full commitment as distinct concepts.

Thus, from here onward, we act as if we are solving for optimal information and incentive constrained multi-agent contracts and trade mechanism by maximizing weighted sums of ex-ante expected utilities

subject to resources and incentive constraints, being clear about whether or not commitment is assumed anywhere in the problem, as well as what is known and unobserved.

Enduring relationships can be helpful, promised utility becomes a key state variable, and notions of time consistency come into play. Promised utility from next period on is a way to write down contracts as if they were two period contracts, contemporaneous for today and the promise as the stand-in for the future. Promised utility is a key state variable that can be used in formulating, characterizing, and computing solutions to contractual problems. One party may wish to withdraw and go their own way, in which case contracts with limited commitment make sure that promised utilities do not fall below certain thresholds, then loaded and recorded into the contract.

In the autarky version of this, there is implicit trust that a banishment penalty can be imposed where withdrawals happen for penalties. In the competitive market version of this, we trust a party not to break the relationship even though it might be mutually advantageous, for example, if one party of a contract is drawn off by a competitor. Relatedly, both parties to a contract may wish to renegotiate and start over, but that would be bad for incentives ex ante. Either the parties trust the former will not happen, as in full commitment models, or a time consistency constraint is appended to the underlying contract problem, so that time consistency is loaded and there is no temptation.

If contracts are incomplete and penalties cannot be imposed internally as part of contract, then we can imagine that promised utilities are tagged to each player as a result of their performance under the contract. This is a way to formalize a role for reputation when there is no trust and a way to design and commit to optimal social penalties.

Finally, one can solve as in mechanism design for constrained efficient ex-ante agreements. Though with assumptions a solution exists and constitutes a valid Nash equilibrium, the solution is not immune to collusion. Thus, either we trust parties not to collude, or additional constraints on the multi-party contracts need to be imposed.

Contracts interface with ledgers and standard financial accounts. An asset as collateral in escrow is not really the same as an individually owned asset, since another party has a contingent claim on the asset and special accounting is required.

It is natural to think of the flow of future wage payments as securitized and creating an item called human capital on the balance sheet. But, when such flows can be collateralized, in this sense, have a contingent claim on them, for example, as collateral for loan repayment, then the modified accounting terminology needs to be created.

## 2.3 Smart Contracts

At its most basic level crypto currencies such as bitcoin use language familiar to accountants, economists, and computer scientists. A flow is currently defined when ownership of a digital asset, (the stock), changes hands, (a change in ownership of the stock). These stocks and flows are recorded on ledgers. Bitcoin verifies and validates flows back to the genesis state where assets were originally created. This tie-in of flows to stocks requires public verification of information.

The concept of ledgers is generalized to mean a list of recorded “facts”. Everyone has a ledger, but the ledger is synchronized only for shared facts and the verification process is defined as consensus broken into two pieces:

1. Validity consensus
2. Unique consensus

Validity consensus is a transition or transaction accepted on the blockchain only when it has the required signatures, both for the current proposed transaction and for every transaction that led up to the proposed transaction. Unique consensus is different and is the key differentiator to make the blockchain attractive. In a unique consensus, a given party is not required to record every transaction, only that necessary to complete a given transaction. However, a party could potentially request missing transaction information from notaries. The latter is necessary to prevent potential problems such as the double spending problem. These transactions and verifications are dependent on the underlying environment and the objective of the ledger, ideally as part of a Constrained-optimal arrangement.

A contract is entered into by multiple parties, with parties defined as nodes on the blockchain. Identities of nodes could be anonymous, as in bitcoin, but this is not required. Identities could be named and public, for example, the legal identifier of an organization or the service identity of a network service. Note that trusted parties such as banks are allowed to be there, and be public, but so are others as well as strangers. A node that writes contracts is an app providing a service, also allowed to be a universal service, if desired. The permissioned set of nodes for a given contract still have their access controlled by a doorman, so in that sense all contacts are only semi-private.

The contract is a code which is initially validated and either works or not (one can imagine several independent validators of code, which links back potentially to thwarting collusion). A contract agreement is made via public and private keys. After this validation, it becomes immutable. In this sense, there is no reneging on whether agreements have been entered into, ruling out claims that the agreement was written in a different way. There is no need for trust at this stage.

A contract specifies states at points in time, such as current ownership for example or other facts. Communication under a contract is node to node, not necessarily broadcast to the entire public, as in Bitcoin, but rather on a need-to-know basis, as pre-specified in the contract. An oracle is used to verify known facts that are states of a contract. Commands initiate transfers and result in output states.

As noted, there is a notary service on an as-needed basis for validation of communication and proposed transactions. Upon being sent a proposed transaction, the notary will either accept it if the notary has not already signed other conflicting transactions, or otherwise will reject it, as would happen in an attempt to double spend. Every state has an appointed notary, and a notary will only notarize a transaction if it is the appointed notary of all the transaction's input states. A notary may be a single network node or a cluster of mutually trusting or mutually distrusting nodes. Notaries can choose a consensus algorithm based on privacy, scalability, legal system compatibility, and algorithmic agility. However, a notary could decide whether to provide validity consensus and if not to run the risk of denial of state attacks.

To summarize, smart contracts are a series of instructions based on technology rather than people or paper. A programming language dictates an IF-THIS-THEN-THAT (IFTTT) logic, in which instructions are only executed when meeting certain criteria. Every step acts like a trigger for the next step to execute itself, creating a domino effect until the contract is fulfilled. The easiest way to envision how smart contracts work is to think of the transaction like a vending machine interaction. Instead of going into a physical store, a person (the requestor) works only with the machine (the provider) to acquire an item without a middleman. Likewise exchange platforms with multiple agents interact with each other through code.

## 2.4 Lessons from Mechanism Design

### 2.4.1 Public vs. Private Information Concerns

We come to the second implication of private information for contract and mechanisms design. Though such private information is effectively reported via messages or indirectly by choice of options, this does not mean that such internal data should be made public on the ledger, even if households were compensated for posting. Often, the opposite is true: messages should be kept private and of course cryptography makes this possible. Ledgers need to be partitioned and permissioned. Further, the platform may deliberately conceal internal information by randomization of observed outcomes. Fortunately, codes have randomization routines built in. A key part of the implementation is the construction of incentive-compatible and optimally-designed information and credit registries

More specifically as in an illustrative economic environment, there are two parties to a contract with varying, random preferences to consume, urgent to consume to get resources today or patient, willing to

give up today in turn for having more tomorrow. There is correlation in these preference shocks over time and over the agents, the parties to a contract. One of the two parties in an initial period can announce its urgency today, getting more goods or value today, or giving up value, depending on the shock, if patient, and the second party announces its preference shock tomorrow. If that announcement by the first agent for today were public, it would undercut insurance possibilities for tomorrow, as more of the underlying unobserved state would be known. The second party wants to be able to smooth tomorrow over its own shocks, if possible, so that insurance is desired ex ante, giving or getting as a function of being patient or urgent. The idea extends to agents suffering from the risk of balance sheet shocks and to multiple agents, as SMEs, money transfer organizations, and the collection of dealers who make markets.

### 2.4.2 Delegation of Portfolios to a Third Party: Platforms as Custodians

Another implication of private information: when there are private unobserved idiosyncratic shocks to endowments, to preferences, or to balance sheets, and as well publicly seen aggregated shocks to technology or asset transformation, then it can make sense for households, firms and traders to delegate portfolio decisions. This can happen in principle with village funds, cooperatives, wealth managers, up to formal exchange-traded funds that can be undone only by designated participants. This allows front loading, giving out more today as opposed to tomorrow. This can happen when incentive constraints bind and limit trade in future periods— the value of having future resources is more limited. But delegation can also be back loading, pushing the rewards to future periods. This strengthens intertemporal incentives by having more at stake in future periods to reward participants, depending on what is said today. These types of incentives here can be undercut if agents (households, money transfer organizations, providers of liquidity) can retain control over part of their portfolio and make decisions on their own. Thus, the argument for having a private e-coin for awards and penalties that by construction are under control of the multi-agent exchange system.

### 2.4.3 Limited Commitment and Default: A Theory based Reputation Score for Contract Participants

The contacts considered above have internal incentives for performance, while ensuring the parties that entered the contract cannot walk away. But default can be incorporated. We can consider, for example, pairwise borrowing/lending contracts in which the borrower can default, i.e. has the option to not pay and would not do so if preferable. However, we can construct a social scoring system composed of promised utilities that serve to reward or penalize good or bad behavior. This then becomes a type of reputation mechanism.

## 2.5 Tokens as a way to Support the Trade, Credit and Insurance

Tokens are a unique way to encourage incentives between groups of people or entities to achieve their part in completing transactions in a trustless network. Traditionally, there is a structure and chain of

command layers each process goes through. Permissionless ledgers (which the EvryNet network uses) omit the need for traditional steps, as follows.

## 2.6 Tokens as a way to Achieve Unique Consensus

Transactions in finance, trade, credit, and insurance require multiple party permissions, per se, to ensure a safe, secure end result for all.

In a traditional transaction, imagine there are four agents in separate locations with subsets of agents. For example, a risk averse Agent A is paired with risk neutral Agent B in one location and these two process the first part of the transaction. Then risk-averse Agent A paired with risk neutral Agent B in the second location. The first part of the transaction needs to be verified. Likewise Agent A initially paired with Agent B, is then paired with Agent B. Only when all four parties reach a consensus is the overall transaction completed. The problems lie with 1) cost — the requestor is paying for four parties and 2) time — the requestor needs to wait for all parties to reach agreement.

## 2.7 Tokens can Solve these Problems

Tokens are considered digital collateral as a tool for people to participate in digital transactions. The intention is a form of payment or incentive for users to initiate specific actions because they would know they will be compensated. Every step of the transaction is recorded on ledgers, enabling actions to be tracked, ensuring all physical and digital assets that hold value can be tokenized and used as collateral. Tokens are reshaping how we develop incentive models to alter human behavior and trust towards performing tasks in more efficient, cost reducing ways.

# 3. OVERVIEW OF EVRYNET FINANCIAL SERVICE ARCHITECTURE

EvryNet financial service platform will be designed using micro-banking services and scalable architecture, incorporating real-world trust, governance and auditability to enable financial services. The EvryNet platform is separated into three primary components:

## 3.1 EvryNet Financial Service Portal

EvryNet platform gives flexibility to transform real-world contracts into smart contracts where they can get created and executed in a fast and economical way. EvryNet financial services can be accessible to SMEs, the underbanked or the general population, through EvryNet Service Providers' portals. Anyone or any organization that has knowledge of composing contracts are welcomed to become an EvryNet Service Provider.

The primary responsibility of EvryNet Service Provider is to create services which utilize EvryNet smart contracts or help users compose or customize smart contracts to meet their needs. EvryNet Service Providers do not require programming expertise. They can compose or customize smart contracts using provided EvryNet SDKs, APIs or templates. EvryNet smart contracts can be composed in two ways: 1) the providers can start from EvryNet smart contract templates and adjust relevant parameters according to actual user demand and 2) the providers can use EvryNet building blocks to create a smart contract that better serves specific needs.

A smart contract consists of transactions which will be executed in order. For each transaction, primitive actions and operations associated with the required financial service need to be specified, such as conditional transfer or refund and other enforceable financial primitives. The operations can take as inputs the associated contractual party accounts and assets. We can define a transaction as a composition of the operations which can then be executed upon a request or certain conditional events. Events indicate internal or external conditions, such as delivery of goods or services and promised actions. To streamline with EvryNet contract network, hooks will be implemented to connect to external systems to feed the events from real, auditable systems providing real-time event streams.
In a smart contract, users can optionally determine the preferences on who should be able to view the contract as well as the minimum number of EvryNet execution nodes. The number of execution nodes will determine the confidence in contract execution as well as the execution fee. Once the smart contract is created, it shall be verified or digitally signed by the relevant contract parties to maintain integrity of the contract.

Additionally, any organization can integrate with EvryNet platforms using EvryNet APIs/SDKs to create specific financial products or seamless services on their platforms. These products or services can be published or discovered through EvryNet network, becoming alternative services for users. Moreover, other financial institutions or companies can offer new services on top of EvryNet network, such as distributed credit rating. EvryNet service providers can utilize EvryNet SDKs/APIs to interact with EvryNet network.

EvryNet platform can offer a verification process for EvryNet service providers in order to increase customers' trust and confidence in particular providers. If an EvryNet service provider passes the EvryNet verification process, the provider will become a verified provider. EvryNet encourages EvryNet service providers to enter the verification process.

In the initial phase, EvryNet platform can support escrow-based smart contracts. To make it concrete, we give examples of smart contract templates to demonstrate the following services:

#### 3.1.1 Letter of Credit

A letter of credit (LC) is issued by a bank on behalf of a buyer to guarantee that the payment will be transferred to the seller once the terms and conditions stated therein have been met. Please visit Section 9.1.1 on Building Blocks for more details on how to compose smart contracts to support LC service.

#### 3.1.2 Bank Guarantee

A bank guarantee (BG) is a contract issued by the bank, wherein the bank gives the guarantee on behalf of the buyer to the seller, that the bank will be responsible for payment if the buyer is unable to pay upon the specified conditions.

#### 3.1.3 Loan

A loan is the lending of money from one entity to another entity at an interest. The borrower will initially receive an amount of money and is obliged to repay the amount at a later time.

### 3.1.4 Supplier Contract Chain

Supplier Contract Chain is one of the waterfall payment problems. In a large-scale procurement or a public project, a contractor usually has multiple suppliers and subcontractors to support the execution of the project. EvryNet platform will provide subcontractors a way to verify the availability of funds from the revenue source and allow enforcement of payment guarantee from the primary contractor. The primary contractor can also verify the performance of the subcontractor before issuing the payment.

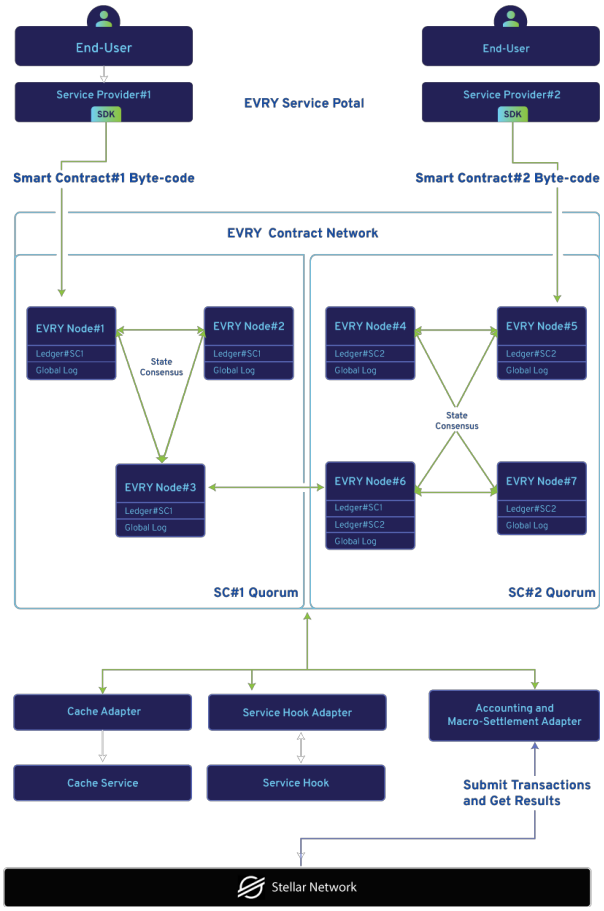


Figure 1: EvryNet Network Architecture

## 3.2 EvryNet Contract Network

The second layer of the EvryNet platform is responsible for executing smart contracts as well as enforcing individual asset settlements upon specified conditions. The overview functionality of the EvryNet network is illustrated in Figure 1. The contract network consists of EvryNet nodes and user nodes or accounts. Each individual node does not need prior trust in one another to initiate business agreements or smart contracts since EvryNet platform will enforce the terms of contract according to what happens. Each node may have its own reputation score which may be reflected from the node's assets, liability, and availability. The score may be used to imply a node's capability of honoring the contracts.

With EvryNet contract network, users can identify their preference in terms of who they allow to see and execute their smart contracts in order to preserve confidentiality and reliability. Users might only want people who are involved in a business deal to review the contracts. One potential problem might be that there might not be a sufficient number of EvryNet nodes to run this specific smart contract among the EvryNet nodes that belong to the involved parties available during the contract execution.

During the execution, the details of the contract will be revealed and processed according to the specified conditions. To mitigate this challenge, users can specify trusted parties for a particular contract, who can be advisors, experts, witnesses or related authorities or institutions like in real-world contracts. Before smart contract execution, the EvryNet platform will select one or more EvryNet node to execute a smart contract according to the user preference and the reputation score of individual EvryNet nodes.

In the smart contract execution, the group of the selected nodes will execute the smart contracts and keep all committed transactions in the individual smart contract ledger, which is kept in a distributed storage. One EvryNet node can execute one or more smart contracts in parallel to achieve scalability, depending on the preference specified in the smart contract. Only limited information on a smart contract and a specific result of a smart contract will be recorded in the EvryNet global log to preserve privacy. Moreover, it is possible that the EvryNet network can also aggregate certain contract transactions from different smart contracts before propagating down to the accounting and settlement layer to perform asset transfers among accounts. With this approach, the details of micro transactions can be obscured.

The EvryNet network serves as a runtime environment for smart contract execution as well as a distributed storage to keep track of smart contract execution. Each smart contract will be executed independently by one or more EvryNet nodes which meet the criteria specified in the contracts. To execute a smart contract, users may want to use one or more EvryNet nodes to process their smart contracts. To minimize an execution fee and to yield the most efficiency, utilizing only one EvryNet node to execute a contract is preferable.

However, if a contract involves many stakeholders that may have different opinions of which nodes to trust, the EvryNet network may create a quorum that contains preferable EvryNet nodes for execution. If eligible EvryNet nodes are more than the minimum required, the reputation scores of each node will be considered. When more than one EvryNet node is required to execute a smart contract, all selected nodes will form a quorum to vote on an execution result. Thus, EvryNet is considering several consensus algorithms, such as Stellar Consensus Protocol (SCP) [6] and Tendermint [7]. It is likely that EvryNet nodes of well-known organizations will be chosen by users to execute their smart contract, thus providing more visibility of smart contract activities to the organizations.

All smart contracts and their execution states are stored in their own contract ledgers which may be kept on EvryNet execution nodes. The information should be encrypted before being written on EvryNet nodes' storage. To provide the contract storage, we consider parts of IPFS [8] to provide an open-source, peer-to-peer file system. IPFS is able to address large amounts of data and provides immutable, permanent, timestamp links to contract files. The EvryNet platform can use this link as a checkpoint in the EvryNet global log.

The more EvryNet nodes join the network, the more capacity EvryNet network will have to store the smart contracts and related execution states. EvryNet nodes may be rewarded with EvryNet tokens in proportion to the storage capacity it provides and the number of completed transactions performed by the node.

EvryNet smart contract network is composed of five key components, as follows:

### 3.2.1 Smart Contract Execution Engine

Each EvryNet node will be running the EvryNet execution engine to process EvryNet smart contracts. First, the engine needs to check that a smart contract is properly signed and not tempered. Once a smart contract's integrity is checked, the engine can start processing the contract, transaction by transaction. If one or more inputs in a transaction is not ready, the execution of this smart contract will be halted until the required inputs are available. When the execution of a smart contract is suspended, the EvryNet execution node needs to record the current states of the execution in the smart contract ledger so the execution node can resume the execution correctly. Once the execution is complete, the engine inserts the contract status into the EvryNet global log. The EvryNet execution engine should be able to process multiple smart contracts in parallel, depending on the EvryNet node's resources.

As shown in Figure 2, each smart contract (SC) has its own chain of information, namely execution states and results. The reason EvryNet has a separate chain for each contract is to achieve scalability and privacy. This way the EvryNet platform can execute many smart contracts more independently, since we eliminate the frequent contention on the smart contract execution ledger. The minimal information regarding a smart contract, e.g. a reference to a smart contract's ledger and the execution status at the end of the execution, will go into the EvryNet global log. Contract detail should be fairly concealed but with sufficient trails for an audit.

As illustrated in Figure 2, each circle represents an execution state. For example, H1,1 is an initial execution state of Smart Contract 1 (SC1) which should contain the smart contract code and the information of all participating parties. If all inputs that are required to execute a smart contract are ready

at the start of execution, then this smart contract will have only 2 circles, i.e. the initial stage and the final stage, as shown in Smart Contract 3 (SC3). In the figure, SC1 and Smart Contract 2 (SC2) get processed at the same time. But SC2 gets halted more often than SC1 because SC2 might depend on many external events to evaluate the contract conditions.

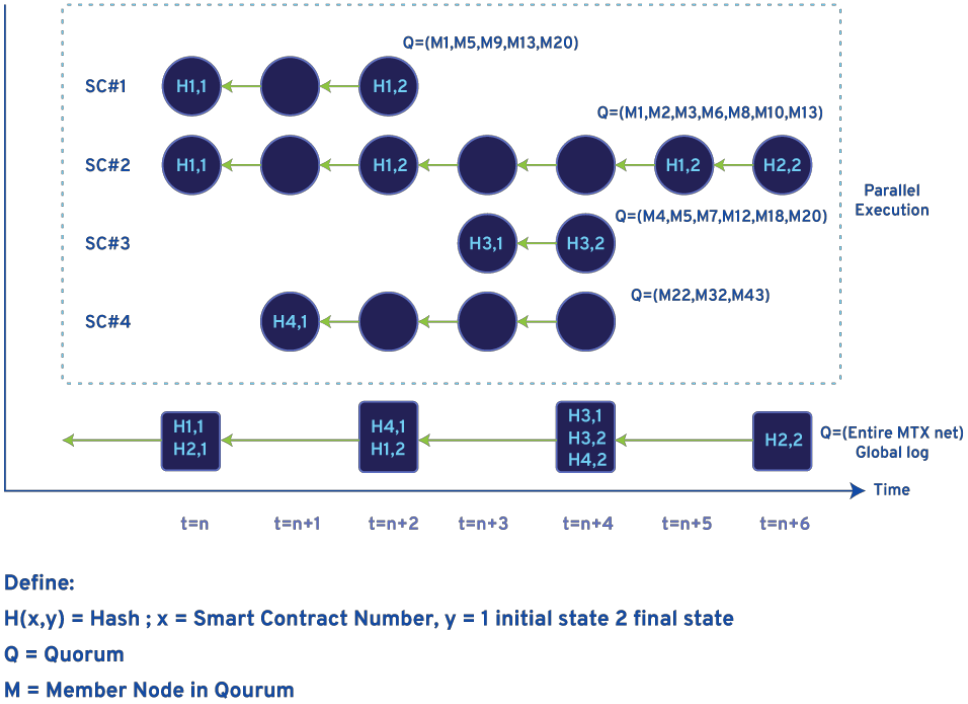


Figure 2: EvryNet Smart Contract Ledger

### 3.2.2 Reputation-based Network

To continuously evaluate the participants in the network, the network maintains a reputation score for each EvryNet node and each user. The reputation score should get updated according to the participants' behaviors which can be implied from smart contract execution. EvryNet nodes are incentivized with EvryNet tokens to contribute resources and execute the given contracts correctly, i.e. honoring the terms specified in the contract and do not tamper with the contract execution.

Reputation component is one of the essential parts in the EvryNet network since EvryNet is open to anyone or any organizations to join as EvryNet nodes. The reputation metric will serve as a tool to penalize ill-behaved EvryNet nodes and reward well-behaved nodes for their good services.

Reputation scores of each individual node or user cannot be transferred as reputation for others but can be aggregated if they are owned by the same organization or person. For example, an organization may have 10 EvryNet nodes participating in the network and one of the nodes can become malicious, resulting

in a bad reputation. However, the reputation of the rest (9 benign EvryNet nodes) are still intact. The reputation of this organization of all 10 nodes would be slightly degraded due to one malicious node.

Reputation metric represents “trustworthiness” of each EvryNet node. More trusted nodes will be more likely to get chosen to participate in smart contract operations. EvryNet determines trustworthiness of a node by the amount of time an EvryNet node spends in a smart contract’s execution and the correct probability of smart contract execution as the indicators. EvryNet prefers a node to process the conditional code as soon as possible without introducing unnecessary delays and of course to execute correctly. The reputation metric of each EvryNet node should be continuously updated, or at the very least, every time a smart contract’s execution is completed.

Our tuition behind utilizing execution time is that if a node takes significantly much more time to execute a similar contract, compared to other nodes, the probability of this node being malicious is higher. Significant delays of a smart contract execution can possibly have an adverse impact when the terms of the contract are time-sensitive. In order to identify these types of malicious nodes, EvryNet compares the execution time of a node to other nodes that have finished execution of the same smart contract class. An outlier node that spends significantly much more time will be considered as a malicious node. For reputation score calculation, see Appendix 1.

#### 3.2.3 Service Hooks

The EvryNet platform also has service hooks which allow EvryNet nodes to register what events a smart contract is waiting to complete a transaction. For instance, a smart contract may have a conditional clause that says, “if the merchandise has not been delivered within 10 days, return payment to the buyer, otherwise send payment to the seller upon delivery.” The contract needs to register with two sources of events: the first source, is the delivery event which could come from an external on-chain delivery service and another source is the timer event (for the 10-day condition). When either event happens, the contract with the event subscription can evaluate the current condition and determine the next step for the execution. EvryNet service providers can also monitor (if allowed by the contract) the execution events of contracts and query external services that generate certain events. For example, when a contract’s execution reaches the point of ‘waiting for delivery event’, the monitoring provider can get the notification from the execution, namely identify the related parties, fetch the delivery status from an external service, and emit the delivery event to the network.

Upon performing certain financial operations in a smart contract, an external state [9] may be required such as credit scores or adjusted liability. Since contract execution might be confidential, the EvryNet platform needs to provide a way for external entities such as a credit bureau, or other rating agencies to interact with the smart contract execution, as shown in Figure 1. The contract can be embedded with

external hooks where an entity can insert the required execution logic (for example to approve or disapprove the transaction), or allow information export (for example the amount of liability incurred by contract execution). Hooks can also be time or event triggers. Smart contract composers can elect to include the hooks provided by external organizations to provide required information, resulting in more extensive smart contracts.

Another example for which EvryNet seeks to use smart contracts is to handle exchanges between two different cryptocurrencies, say XLM and BTC, called atomic swap transactions. To facilitate this example, the EvryNet platform would need an external broker service to provide a recommendation on an estimated fee for execution on both chains. It is possible that a Bitcoin transaction can stay pending or unconfirmed for a long period of time or forever. One primary responsibility of this broker service could be collecting statistics on the confirmation time of a Bitcoin transaction and an associated fee, then the service can generate a probabilistic fee suggestion when submitting such an atomic-swap smart contract. The broker service should recommend an appropriate fee for the Bitcoin transaction that has higher probability to be selected by the miners, to be processed during a reasonable period of time.

### 3.2.4 EvryNet Caching Service

Caching service is created to accelerate different queries and turn them into simple look-ups. One example of queries can be finding a reputation number of certain nodes or users. Another example is listing all smart contracts owned by a user or listing the smart contracts that an EvryNet node has executed.

### 3.2.5 Adapter Service

The adapter service gives EvryNet the ability to build on various supporting technologies, such as distributed storage systems, blockchain-based asset transfer, event notifications, and other existing technologies. The adapter service allows the core EvryNet services to expand to different technologies without reimplementing the entire system.

## 3.3 Accounting and Settlement

The accounting and settlement component should provide the final settlement network where assets are exchanged in real time among relevant accounts, as well as a non-reputable ledger of actual asset transfers. Two key properties of this component are as follows:

### 3.3.1 Asset/Liability Tracking and Accountability

The ability to trace all asset movements associated with each account at any moment on a distributed global ledger. In addition to the assets, EvryNet also uses the distributed global ledger to track account

liability expressed in the form of account attributes, such as credit-rating, liability, and other properties related to the transactions. This allows transparency in audit and traceability if necessary.

### 3.3.2 Fast and Affordable Asset Exchange

The ability to transfer real-world assets or their digital twins using cryptocurrency should be within a matter of seconds with minimal fee. EvryNet is considering leveraging the Stellar network [10] to provide the accounting and settlement component in a fast and affordable manner. Stellar has a capability to settle, track, and transfer assets which, in the real-world, could be currency. One of upcoming features on Stellar is the ability to perform cross-platform exchanges. Stellar should be able to handle all the last-hop financial transactions, such as actual asset transfers. Stellar real-time settlement and low-cost transaction fees, allowing EvryNet to implement sophisticated financial services on top of its network. Figure 1 illustrates how EvryNet network would interact with the Stellar network.

# 4. EvryNet Smart Contract

## 4.1 EvryNet Smart Contract Characteristics

An EvryNet smart contract can be viewed as a set of ordered transactions. Each transaction needs to be executed in a specified order. If one transaction with a smart contract fails, the smart contract will fail immediately. The composer can specify a finite number of retries to handle transient problems, such as temporary network or service congestion. Retry time could be modeled, see Appendix 2 for smart contract objects.

## 4.2 EvryNet Smart Contract Life Cycles

### 4.2.1 Creation (EvryNet Service Provider) Composition

The EvryNet Platform provides an SDK for EvryNet service providers to compose, compile, link and validate a smart contract. EvryNet service providers can easily configure EvryNet smart contract templates to satisfy users' terms and conditions without programming or coding knowledge. Alternatively, EvryNet service providers can simply combine EvryNet building blocks and adjust parameters to customize individual smart contracts.

### 4.2.2 Compilation/Linking

EvryNet service providers can use the EvryNet SDK to compile high-level smart contract specifications into a smart contract executable. In order to generate the executable, the given smart contract specification needs to compile and resolve all references which can be other smart contracts, compliances, rules/regulations, or external events. For compliances, rules or regulations, the compiler needs to contact EvryNet Hook Service to return the links to the most updated versions of particular compliances, rules or regulations, and update the smart contract accordingly. The third parties who provide compliance services or regulators need to connect with the EvryNet Hook Service when a new version is available.

A smart contract dependency graph will be created to handle the references among smart contracts or modified if the graph already exists. The parent of the node is the smart contract that the child node depends on. The root of the smart contract dependency graph is always the topmost smart contract that all children smart contracts depend on. The smart contract dependency graph allows EvryNet to mimic real-world contracts where one contract leverages on an expected value of another contract. For example, a smart contract may refer to another smart contract for collateral. It is necessary to ensure the referenced contract exists, and the user has the right to leverage such a contract in addition to checking the value of the collateral contract.

To resolve external events, the EvryNet service provider needs to communicate with EvryNet Hook Service to facilitate an event resolving process. For instance, when a smart contract requires a delivery notification from Thai Post Office Service to complete the condition, the EvryNet compiler will register this notification event with the EvryNet Hook Service and update the smart contract accordingly.

#### 4.2.3 Digitally Signed

To complete a smart contract creation, EvryNet service provider needs to have all related parties digitally signed to ensure the contract integrity, including the provider. The signed smart contract file will be encrypted and pushed to the underlying distributed storage layer where this file will be turned into an immutable object. The key that is used to encrypt a contract can be a symmetric encryption key shared with all involved parties of this contract.

Then, the EvryNet service provider can submit a request containing the immutable contract link to the EvryNet request queue in the EvryNet platform. The request will have the link of the written persistent smart contract object (e.g. IPFS object link) and the fee to execute the contract. Fees will be paid in EvryNet tokens. If users specify a particular EvryNet node to execute their smart contracts, the request will directly go to the specified EvryNet node's queue.

Moreover, the severity level can be tuned to allow EvryNet platform to optimize the processing time and associated fee. For instance, if a smart contract contains minimal risk and privacy, then EvryNet will select any available EvryNet nodes with an acceptable reputation to process a contract.

#### 4.2.4 Execution (EvryNet Contract Network)

In the execution phase, an EvryNet node will pull a smart contract request from either their own queue or the EvryNet global request queue. If this request is new and has not been executed yet and the preference says one execution node is acceptable, then the assigned EvryNet node can start executing the smart contract immediately. However, if a smart contract requires more than one EvryNet nodes to execute, then a quorum of the required EvryNet nodes needs to be formed first. The more EvryNet nodes can execute a smart contract, the more chance the smart contract can be processed quickly since EvryNet nodes can be temporarily busy or offline.

If a smart contract's execution needs to wait for a certain event to happen, the execution of the smart contract will be halted until the event is fired. Examples of events can be certain date/time, expiration date, shipment tracking number from users, delivery status from a shipping carrier, execution of another smart contract, and the confirmation/result from Stellar when asked to perform a financial transaction.

When a smart contract waits for a certain event, its status will be pending. The pending smart contract will be resumed when the event is triggered. When execution of a transaction fails due to the missing of any specific external events, EvryNet can keep retrying the transaction if the number of retries is not exceeding the maximum retry threshold. The retry mechanism is necessary to cope with transient failure, especially when dealing with external entities. However, EvryNet also imposes the maximum number of retries to prevent a DoS attack. Once all transactions in a smart contract are successfully processed, the smart contract will enter the final settlement stage.

# 5. EVRYNET'S BUILDING BLOCKS FOR SMART CONTRACT CREATION

The EvryNet Platform provides SDKs and templates to help EvryNet Service Providers to compose or customize a smart contract easily. A smart contract can be created using our fundamental building blocks defined as follows:

## 5.1 Contract

EvryNet contract is a composable object. These objects consist of values and code which are executed to verify or redeem additional values upon certain conditions or external events. The EvryNet contract can subscribe to external events which are required to satisfy the given conditions. If the conditions are met, the contract execution will advance its state and alter its relevant values.

## 5.2 Asset

Assets are used to substitute currency that the intermediary or the seller can claim instead of monetary value. Assets can either be a representation of values (such as money) or another smart contract (which can be redeemed as monetary values upon certain conditions).

## 5.3 Condition

Conditions are composable logics which are used to determine which action to take based on the contract code and external events.

## 5.4 Event

External events that can be published to the EvryNet network by external entities. Events, such as timing, delivery of real-world merchandise or services, can trigger a smart contract execution, which eventually leads to payment or transfer of values.

## 5.5 Action

Action is an operation that must occur when the condition is satisfied.

# 6. SMART CONTRACT FEE

When an EvryNet Service Provider submits a smart contract for execution, the contract must be shipped with the fee to get accepted in the network. The execution fee will consider the following:

- The complexity of the smart contract (e.g. the number of transactions, references to other contracts and usage of hook services)

- The execution quorum (e.g. better reputation node may require higher fee and more nodes can result in higher fee)

The execution fee can be paid in EvryNet tokens which will be distributed to the following parties:

- EvryNet Nodes: Nodes that execute a smart contract

- EvryNet Platform Extension: Hook services / External event services; Stellar network which serves as a settlement network

- EvryNet Governance: Based on the governance policy of the network, a small portion of the fee will be collected into a central pool and redistributed according to the governance policy to incentivize network participation.

# 7. EVRYNET TOKENS

EvryNet tokens (EVRY) serve as the primary mechanism to incentivize and attract constituents to utilize the network. The tokens allow financial service providers to access EvryNet services and are used as gasses to compensate infrastructure providers. The tokens are also used to incentivize users to provide relevant financial data for service providers and can also be used as credit rating enhancement tools.

## 7.1 Token Utility

### 7.1.1 Membership accesses

Financial operators and service providers must maintain and stake/deposit a certain amount of EVRY tokens to earn the right to operate on the EvryNet network. This grants access to the EvryNet execution engine as well as its interoperability to the banking, regulatory, reputation (credit rating) infrastructures required to provide services to their customers. Increases in the participation of the operators and service providers on the network will be constructive to the price of EVRY tokens.

### 7.1.2 Gases for transactions

EVRY tokens is also the fuel for the network. The gas fee is designed to reimburse the EvryNet network for the cost of transactions incurred on a per-transaction basis, as well as to prevent malicious attacks on the network. In the case if EVRY tokens are priced too high (when token price rises), the value of the gas fee will adjust back to the level required to sustain the network's competitiveness. The increase in the usage of the network on a per-transaction basis will cause an increase in the price of the token.

As part of a contract's specification, contract composers can specify and limit the amount of gas used in the execution and the qualifications of nodes that can execute the contract. The number of execution steps can also be derived from the content of the contract. Before execution, different providers can offer a range of fees for the contract. The EvryNet network can then match the transaction with the service provider whose fee is within the limit specified in the contract. After the completion of each operation of the contract, the transaction fee can then be transferred to the chosen provider. This market-based mechanism will allow the gas value to be adjusted according to the demand of the network usage.

### 7.1.3 Rewards

The individual users and entities that are important constructs of the EvryNet ecosystem, are rewarded and incentivized with EVRY tokens for sharing their financial data to the EvryNet network. The financial data is an essential part of the EvryNet reputation functions that help construct the EvryNet's proprietary credit rating capacity. The EvryNet reputation network is an important tool for financial operators and service providers to gain access to an optimal risk-adjusted return environment when they issue financial products (i.e. loans underwriting, insurances, hire purchase products) to their customers.

### 7.1.4 Credit rating enhancement tools

Credit rating enhancement is a mechanism whereby the participants of the EvryNet network attempt to improve their debt or credit worthiness required to perform a certain transaction, i.e. auction, project bidding, or to obtain better terms for a certain transaction, i.e. interest rate on loan, debt issuance. Through credit rating enhancement, there is reassurance that the counterparty of a transaction will honor its obligation through additional collateral, insurance, or a third-party guarantee provided.

## 7.2 Token Flow

EVRY tokens are used to drive the adoption and provide incentives for participants on the EvryNet network. Figure 3 shows the interaction among EvryNet network components and how EVRY tokens are utilized in each steps of a smart contract execution.
EvryNet App must deposit a certain amount of EVRY tokens in order to initiate an execution of a smart contract. A portion of the deposited EVRY tokens will be used as gas, which will be distributed to all relevant parties who contribute to the smart contract execution. The other portion of the deposited tokens will be considered as stake. If the EvryNet App misbehaves, its stake will be seized as a penalty, otherwise, its stake will be left untouched. This mechanism discourages superfluous transactions and alleviates a denial-of-service attack or any malicious attempts on the network.

A gas portion of deposit tokens will be distributed to relevant EvryNet components as follows:

- EvryNet Nodes which participate in a contract execution will receive the tokens to compensate for their services. This includes validator nodes and executors which execute a smart contract on behalf of users.

- If a transaction involves a transfer of assets on the Stellar network, a gas portion of the deposit tokens will be used to pay for the Stellar transaction fee (in XLM).

- External services including hooks or oracle nodes which provide connectivity to services outside of the EvryNet network will also be compensated using the deposit tokens.

- A portion of the tokens will be deposited into the EvryNet reserve pool and will be distributed to EvryNet network participants based on their contributions to the network; including

    - App Owners will receive EVRY tokens based on the number of users who utilize the App to compose and submit smart contracts. This incentivizes the App owner to provide competitive services to attract users.

- Token holders will receive EVRY tokens based on the value of tokens in their account. This incentivize users to seek and hold EVRY tokens as an alternative saving.

- EvryNet users who opt to provide the information regarding their contract execution or behavioral data will be awarded with EVRY tokens as compensation for giving out their data. The data may be anonymized to preserve privacy but still adequate to perform data analytics, such as predict market trends or establishing a baseline for a credit rating. The data could also be supplied to get personalized services.

- User agents who serves as an aggregator for multiple users (e.g. a village fund manager) will also receive EVRY tokens to incentivize the agent to promote token usage.

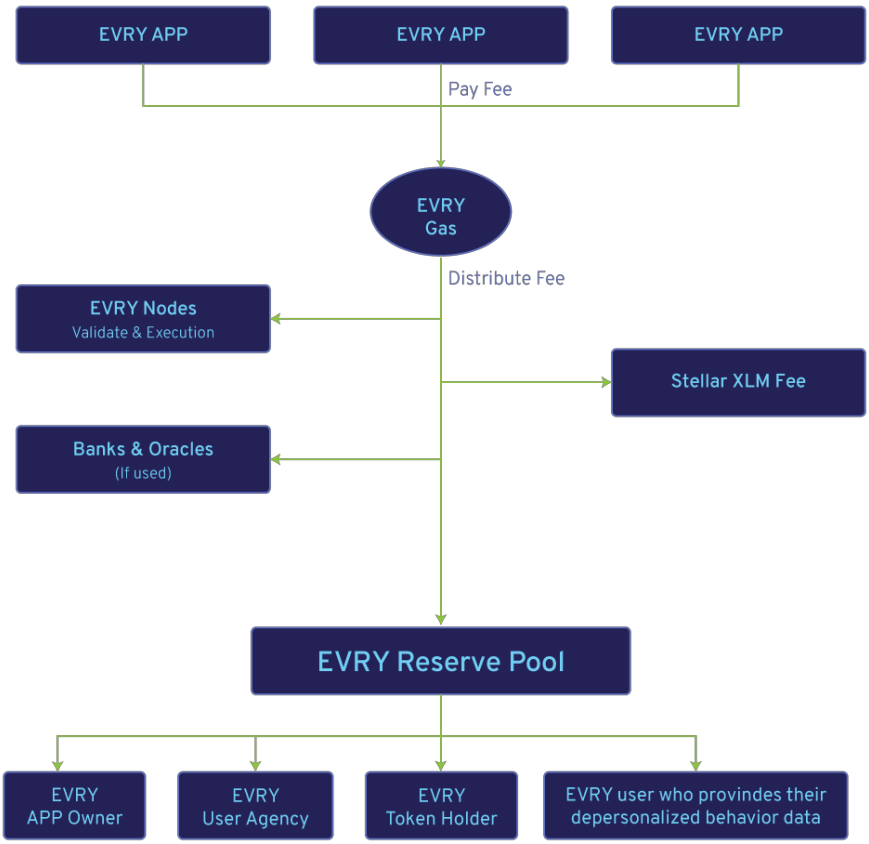


Figure 3: EvryNet Token Flow

# 8. GOVERNANCE

EvryNet Platform will be governed by the EvryNet governance council. The council should include EvryNet stakeholders who will be impacted when major changes happen on the EvryNet Platform. EvryNet stakeholders can be the following individuals or organizations:

- EvryNet users
  Major users: Top x% smart contract owners (high volume)
  Minor users: Issue y smart contract per period (low volume but consistently use the platform

- EvryNet token holders must have at least z EVRY tokens

- EvryNet nodes need to currently or have participated in smart contract execution (either contribute computing power or storage capacity) and their reputation metrics must not be below an acceptable threshold

The EvryNet governance council will be assembled at the time of voting so members of the governance council can change from one voting to another voting. Each member has an equal vote. The EvryNet governance body will have complete control over the protocol. The control includes the ability to manage exceptional events including protocol upgrades, fixes, setting default thresholds for consensus and other network-related parameters. Token holders will be able to vote for proposed updates and key functionalities to the EvryNet network.

# 9. PROJECT MILESTONES

## 9.1 Phase I

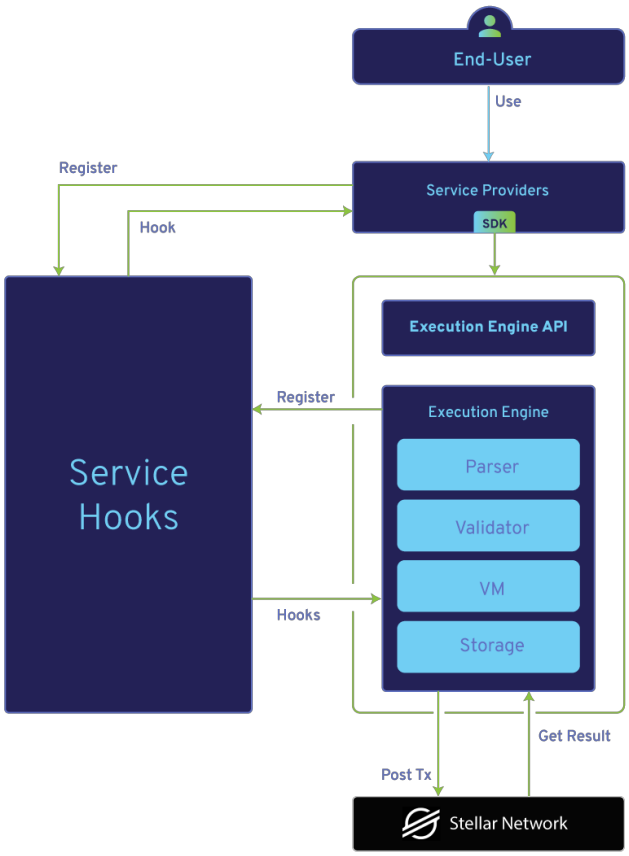


Figure 4: EvryNet Network Phase I Architecture

In this phase, we focus on smart contracts which includes six primary components as follows:

### 9.1.1 EvryNet Smart Contract Building Blocks

The EvryNet building blocks are provided to EvryNet Financial Service Providers to easily create more customized smart contracts for end users. EvryNet Financial Service Providers do not necessarily have excellent coding skills since the provided building blocks resemble pieces of regular contracts. Smart contract building blocks can be as follows:

- Contract may have a finite set of deterministic states and conditional transitions with termination property.
- Assets are digitized versions of real-world assets, which can be acclaimed in monetary value. This can include our EvryNet smart contract itself.

- Conditional logic that may evaluate assets or events according to certain criteria and will determine one or more of the following actions:
- Events may represent something that happens internally or externally. Events can be used as a trigger for condition checking.

Action is a defined operation which may affect the outcome of the contract.

### 9.1.2 EvryNet Smart Contract Primitive

EvryNet primitives are defined operations which are permitted in EvryNet smart contracts. We plan to investigate a set of primitives from existing works [11] which can be used to describe a wide variety of financial contracts. For example, one can start from a no-obligation contract, acquiring currency units, assigning rights and obligations, or specifying conditions when a given contract should be acquired by another party based on given conditions or observable events.

### 9.1.3 EvryNet Smart Contract SDK

EvryNet smart contract language will build upon a set of primitives to resemble a regular contract. Contract composers can combine multiple contracts to create financial instruments that can be executed in the EvryNet network. The high-level language will be compiled to an intermediate representation that is executed across the network or parties running EvryNet execution engines. Using the language, users should be able to specify information related to the contracts including acquisition date, observables (from events), scales, options, limits and how to abandon or terminate a contract.

### 9.1.4 EvryNet Smart Contract Intermediate Representation

Intermediate representation (IR) is necessary for validating the smart contract. The smart contract needs to satisfy at least one of the following three properties to be allowed to run in EvryNet network:

- Finite
- Deterministic
- Termination

IR will be constructed from the smart contract which may be composed from EvryNet templates or EvryNet Building Blocks.

### 9.1.5 EvryNet Smart Contract Templates

EvryNet templates are designed to help EvryNet Financial Service Providers to compose common real-world contracts, such as, contracts for online retail, waterfall contract and letter-of-credit-like contract. The

templates are essentially created from EvryNet Smart Contract Building Blocks in the order that they resemble such real-world contracts.

### 9.1.6 Interface

**Interface for EvryNet Smart Contract Providers**

EvryNet Smart Contract Providers can utilize EvryNet smart contract services through EvryNet SDKs and query information or submit the smart contract via EvryNet APIs. We will design and implement EvryNet SDKs to facilitate the use cases defined in Section 3.1. APIs for information query as well as smart contract submission will also be defined. Examples of information query can be 1) to show a list of available external events to use in a smart contract, 2) to show a list of verified external partners which may supply external events, 3) to show the current execution status of a smart contract, and 4) to show the information of a smart contract, e.g. owners, involved parties, dependent smart contracts and so on.

**Interface for Event Providers**

Event Providers are entities that can generate specific events needed in a smart contract. Examples of event providers can be banks, delivery service providers, and system processes that represent timers. We will define APIs for query, update, subscribe and unsubscribe events that smart contracts may need from Event Providers.

### 9.1.7 Runtime environment for Execution Engine

In the EvryNet execution engine, a runtime environment is a critical part for success of EvryNet smart contract execution. An implementation of EvryNet reference runtime should be available for any service providers or organizations that would like to join the EvryNet contract network. We will work on the stability and security of the EvryNet runtime while ensuring the auditability and validity of contract execution. We will implement a proof-of-concept of the EvryNet Smart Contract Execution Engine. Initially, EvryNet or verified partners may be the ones to start deploying the first implementation.

## 9.2 Phase II

In Phase II, we concentrate on the design and implementation of EvryNet Contract Network in Section 3.2. We will leverage EvryNet nodes deployed in Phase I to form EvryNet network in a distributed manner for scalability.

## 9.3 Phase III

In Phase III, we will focus on the model and algorithm of the EvryNet reputation network.

# 10. USE CASES

## Evry.finance

Evry.finance is a financial application to be built on EvryNet. We present two target groups as use cases. One case for institutional investors looking to participate in the decentralized finance (DeFi) and the second case, for small retail investors. Both groups are looking to invest in higher returns while diversifying in portfolios of assets, avoiding traditional dealer markets and limited options. Both cases overlap in the sense that technology developed for one can also be applied to the other. We present each to make the point that the EvryNet platforms can support diverse, seemingly different applications.

## 10.1 Hybrid DEX with AMM Pool and Order Book

The exchange in question here is exchange of crypto-currencies, stable coins, and fiat, though the technology being developed will allow exchange in other objects, e.g. household and SME debt as above. This segment of the market is receiving more attention recently, so we elaborate on the connection to EvryNet and how it differs from trusted third party dealers.

The Decentralized exchange allows users to trade one token for another on EvryNet, in a non-custodial manner. A User's order may be matched by either an order book, or the automated market maker pool (AMM Pool The AMM pool is a better choice for tokens with low trading volume, while an order book is better for large orders as it has a low price impact, if the order book is active.. But the program will be coded to automatically choose the best option for the client.

## 10.2 Liquidity Pool (Yield Farming)

The liquidity pool is an integral part of the AMM Pool DEX. It is a pool with two (or more) tokens available for users (traders) to trade without the presence of a counterparty. If the trader is a buyer, then the liquidity pool acts as a seller. On the other hand, if the trader is a seller, then the liquidity pool acts as a buyer. Trading fees are charged to the trader and distributed as profit to liquidity providers. Liquidity pools offer an opportunity for users with inactive cryptocurrencies to get variable yield by providing liquidity into the liquidity pool.

- 

## 10.3 Lending & Borrowing

DeFi users can trade on margin or take a short position by borrowing funds from lending pools, while suppliers of tokens to the lending pools earn interest from lending. Liquidity providers to the liquidity pool also earn some interest by allowing a portion of the pool to be lent out. Leverage yield farming can be

manually done by borrowing from lending pools with cheaper interest and then providing liquidity in the liquidity pool with higher yield.

## 10.4 Stablecoin Minting and Integration with VELO Protocol

In addition to borrowing stablecoin from the lending pool, users can pledge their tokens as collateral and mint new stablecoin to use in their trading. Instead of minting USD-denominated stablecoin, with VELO Protocol integration, users can also choose to mint stablecoins of other foreign currencies. This will enable Non-US users to hedge their portfolio to minimize FX risk and trade in diverse coin markets.

## 10.5 Synthetic Asset

Users can pledge their tokens as collateral and mint a new synthetic asset, which has a real-world underlying asset such as commodities, stocks, indices, or real-estates, and get exposure of these underlying assets on DeFi without selling and withdrawing their crypto to buy these underlying assets. The Synthetic asset can be traded on the DEX and can be provided as liquidity to the liquidity pool. A secondary market for asset-backed tokens, i.e. tokens with funds raised from asset tokenization deals, is also available for users to trade on.

## 10.6 Fixed-interest Rate Protocol

For select tokens with high liquidity, users can invest their tokens into the lending pool to get a fixed interest rate. This can be made by offering multi-levels of interest to multiple tiers of users based on their preference (fixed vs floating interest). Users that invest their tokens in the fixed-interest lending pool will get a fixed, but lower, interest rate which is more suitable for institutional investors, while another group of users investing their tokens in the floating-interest lending pool will leave their upside open for higher interest rate if there is more borrowing volume than expected.

## 10.7 Portfolio Management & Analysis

Portfolio summary, management, and analysis, are all provided in one page, for easy monitoring and asset management. Analytics on risk and return are also provided.

## 10.8 Mutual Fund & Index Fund

A Trader (fund manager) with exceptional portfolio management performance can set up a mutual fund and allow other users to join in the investing, with a fund management fee charged based on profit sharing. Index funds can also be set up with automatically rebalanced portfolios based on specified

weight of index constituents, without the need of a fund manager. This creates new opportunities for both retail and institutional investors to invest in mutual and index funds in a decentralized manner.

## 10.9 Futures & Options

Users can buy options by paying option premiums to the liquidity pool and have the liquidity pool as an options writer. Users can also trade perpetual futures (both long and short) against the liquidity pool of highly-liquid tokens.

## 10.10 Dedicated Section for Institutional Investor

Institutional investors can explore investment opportunities in highly-liquid crypto assets and financial products in a dedicated section, for easy navigation without searching for liquid assets throughout the DeFi App by themselves. An Institutional wallet with multisig is natively supported on the platform.

## 10.11 Advance Financial Products

Total return swap, convertible perpetual security, and other advanced financial products, will be available in Phase 5. This will close the gap between the traditional financial market and DeFi.

## 10.12 Launchpad

Initial offering of new tokens on EvryNet.

## 10.13 Future Development

Once Evry.finance has enough monthly active users, we may introduce new features such as support for NFT, interactive gaming with focus on financial learning, and other advanced financial products, subject to future demand on each product.

# 11. ACKNOWLEDGEMENTS

Thanks to Jed McCaleb and David Mazières for input and feedback.

# 12. REFERENCES

[1] Nick Szabo. 1997. Formalizing and Securing Relationships on Public Networks. http://journals.uic.edu/ojs/index.php/fm/article/view/548/469/.

[2] Thomas Bertani. 2016.Proof of Identity on Ethereum (or the “KYC problem”). https://blog.oraclize.it/proof-of-identity-on-ethereum-or-the-kyc-problem-f4a9ee40af21/.

[3] Hyperledger Fabric. Channels. https://hyperledger-fabric.readthedocs.io/en/release-1.2/channels.html/.

[4] Thomas Bertani. 2016. Understanding oracles. https://blog.oraclize.it/understanding-oracles-99055c9c9f7b/.

[5] Jules Dourlens. 2017. Oracles: bringing data to the blockchain. https://ethereumdev.io/oracles-getting-data-inside-blockchain/.

[6] David Mazières. 2016. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus. https://www.stellar.org/papers/stellar-consensus-protocol.pdf/.

[7] Jae Kwon. 2014. Tendermint: Consensus without Mining. http://tendermint.com/docs/tendermint.pdf/.

[8] Juan Benet. 2014. IPFS - Content Addressed, Versioned, P2P File System. https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf/.

[9] Bitcoin Wiki. Using external state. https://en.bitcoin.it/wiki/Contract#Example_4:_Using_external_state/.

[10] Stellar. Stellar Network. https://www.stellar.org/.

[11] S.L. Peyton Jones and J-M. Eber. 2000. “Composing contracts: an adventure in financial engineering,” Proceedings of International Conference on Functional Programming, Montreal, 2000.

# 13. APPENDIX 1: REPUTATION SCORE CALCULATION

To calculate a reputation score, EvryNet first measures the amount of time that an EvryNet node uses to execute a smart contract until it finishes processing a smart contract, excluding halting time or the time it takes to wait for external events. Assuming a smart contract class, C, has n smart contracts, which are M1, M2,M3,…, Mn. Note that n smart contracts may be executed by different EvryNet nodes.

$$Q = \{M_1, M_2, M_3, \dots, M_n\} Eq(1)$$

Each smart contract in the class C, needs an amount of time, tMi , to complete execution with an execution result, VMi . We compute the Euclidean distance of smart-contract-execution time. As seen in Eq(2), we calculate the sum of square distance (d) between each time pair. We then apply the softmax function to distinguish a smart contract executed by an EvryNet node with peak distance in Eq(3). The resulting time score sMi then has a value of range between 0 to 1. Note that most smart contracts in the same class should have execution time clustered around the average value, assuming that there are more well-behaved nodes than malicious ones. The majority of smart contracts should have the time score close to 1.

$$d_{M_i} = \sum_{j=1}^{n} (t_{M_i} - t_{M_j})^2 \; Eq(2)$$

$$s_{M_i} = 1 - \frac{e^{d_{M_i}}}{\sum_{j=1}^{n} e^{d_{M_j}}} Eq(3)$$

The second part of the reputation score is the correctness probability of execution of a smart contract Mi , denoted as pMi . If users are fine with one EvryNet node processing their smart contracts since they already trust the owners of the EvryNet node, then the correctness probability will be one. EvryNet needs to compute the correctness probability only if there are more than one EvryNet nodes participating in a smart contract's execution. To calculate the correctness probability, EvryNet computes the ratio of the

number of EvryNet nodes that have the same vote and the total number of EvryNet nodes that execute a smart contract. By generalizing that VMi is either 0 or 1, the probability can be calculated using Eq(4).

$$p_{M_i} = 1 - \frac{\sum_{j=1}^{n} |V_{M_i} - V_{M_j}|}{n} Eq(4)$$

After completing a smart contract's execution, the reputation score of each participating EvryNet node will be updated based on the combination of pMi and sMi and the previous value of RMi in Eq(5). RMi implies the likelihood that Mi will be able to correctly execute a smart contract in a timely manner. EvryNet sets the initial value of RMi to 0.5 for new nodes. Since the reputation is accumulative, we can adjust the learning rate (α) to determine the impact of the latest execution result to the overall reputation of the node. Note that EvryNet's reputation system will favor nodes with a majority vote and takes minimal amounts of time to execute.

$$R_{M_i}(t) = (1 - \alpha) \cdot R_{M_i}(t-1) + \alpha \cdot s_{M_i} \cdot p_{M_i} Eq(5)$$

At the end of every smart contract execution (finish or abort status), the reputation of all participating users RUi , EvryNet providers and EvryNet nodes will be calculated using Eq (6). We will update the value of RUi based on the final status of the contract. For example, if a contract has been successfully executed, the value of R will be 1. If the user who is specified in the agreement defaults on the contract (e.g. failed to fulfil one or more obligations specified in the contract), the value of R will be 0. Note that in these cases, the learning rate (α) can be higher than the rate in Eq(5) as the recent behavior has a higher impact on the reputation of the user. The reputation value of EvryNet providers can also be computed similarly, for example, invalid or ill-written contract will result in R = 0, and for EvryNet nodes, if runtime or calculation errors happen, the value of R will be 0.

$$R_{U_i}(t) = (1 - \alpha) \cdot R_{U_i}(t-1) + \alpha \cdot REq(6)$$

# 14. APPENDIX 2: EVRYNET OBJECTS

**Smart Contract Object**

A smart contract in the EvryNet platform will be represented as an object of SC type. All SC objects are backed by persistent objects on distributed storage, e.g. IPFS object. Hence, when SC object is modified, a new version of the persistent object will be created. IPFS has the version control system to track the changes in the smart contract object.

- Attribute
  - Timestamp
  - List of all owners (user objects) List of required hooks
  - Trust circle (specify the subset of trusted EvryNet nodes) Privacy level (determine what nodes to be selected)
  - Severity Level (use to use writeback/writethrough optimization, e.g. wait until all layers to confirm before moving to the next transaction when a contract needs to be very sure).
  - Current stage of smart contract
    - Creation
    - Compilation/Linking o Validation Validation
    - Execution
  - Final Settlement Link to current transaction
  - Current execution status – Each stage can be in different status as follows:
    - Success: The current transaction is successfully executed.
    - Pending: The smart contract is waiting for an event in order to complete the current transaction.
    - Ready: The smart contract is ready to be executed. This means it does not wait for any event to execute the next instruction.
    - Retry: The previous execution of the current transaction has been failed. EvryNet is retrying the current transaction again.
    - Fail: The execution of current transaction is failed.
    - Abort: Users send the abort command to cancel the smart contract operation.

- Action
    - Link to the persistent smart contract object that is digitally-signed upon the completion at the creation stage. This object will be persistently stored on a distributed file system (it can be IPFS link to the immutable smart contract file which contains conditional code)
    - Link to smart contract dependency graph List of quorum members
    - Compile
    - Link
    - Validate
    - Execute
    - Retry
    - Show-Current-Status: Show the current stage and status of the smart contract o View-Log: List all transactions and associated status/results of the contract
    - View-Log: List all transactions and associated status/results of the contract List-Quorum: List all EvryNet nodes that are selected in the quorum
    - List-Attribute Final-Settlement

**Event Object**

An event object serves as an evidence of an external event or input that happens inside the EvryNet network (such as particular contract execution) and outside the EvryNet network (such as expiration date, or delivery of goods or services.) SC objects can subscribe to an event and perform associated tasks, including evaluate the next execution step or emit other events for other contracts to perform. Events will be recorded on the EvryNet global log, so it can be used to verify the validity of contract execution.

- Attribute
    - Timestamp: Indicating when the event has been generated
    - List of topics for event subscriptions: EvryNet service hooks subscribing to the event topics will be notified and continue its execution.
    - List of Key-Value pairs: contain the information relating to the event.

**Asset Object**

Asset represents something that a user owns. Asset can incur value or liability.

- Attribute
    - Owner
- Action
    - Value: Calculate value of the asset
    - Liability: Calculate liability of the asset

**User Object**

- Attribute
  - Reputation
- Action
  - Order: Request an EvryNet service provider to create a smart contract according to the specific terms and conditions.
  - Abort: Require the referenced link to the specific smart contract that user wants to cancel
  - Show-Current-Contract: List all current smart contracts that are still under processing with the associated status
  - Show-All-Contract: List all smart contracts that the user has been ordered, associated with the status
  - Show-Attribute: List the values of each attribute o Asset: List all assets that this user owns
  - Liability: Calculate liability of the user
  - Creditability: Calculate reliability of the user
  - Reputation: Calculate reputation of the user

**EvryNet Node Object**

- Attribute
  - Owner
  - Reputation
- Action
  - Reliability: Calculate reliability of the node
  - Availability: Calculate availability of the node
  - Stagnant: Calculate how stagnant the node is

# 15. APPENDIX 3: SAMPLE CONTRACTS

**Letter of Credit**

A bank can issue a letter of credit (LC) on behalf of a buyer to state that the payment will be made to the seller once the obligations stipulated under the contract have been fulfilled. The LC smart contract can be a simple escrow contract where the buyer's payment will be held in an escrow account. The payment can only be transferred if and only if the obligations are met. The obligations are translated in terms of conditional clauses in the smart contract which could be hooked with the external events, such as the electronic notification of the shipment and the interrupt timer to watch delivery deadlines. For example, if the seller cannot satisfy the conditions within the given period of time, the money in the escrow account will be automatically transferred back to the buyer's account. Otherwise, the escrowed money will be made to the seller once all conditions are fulfilled. Here is a pseudo smart contract for LC service.

**Code Example**

```
contract ECommerce(start=time(06/20/2018), end=time(06/29/2018)) {
        //constructor to initialize contract paramters
        def init(buyer, seller: address, amount: num, asset: asset<BTC>, event: event<url>) {
                this.escrow = new Escrow(signers=[buyer, seller]) this.escrow.Lock(asset=asset,
                amount=amount, from=buyer) this.data : MessageQ
        }
        def main() {
                // verify will block until the condition has been satisfied
                // assuming event.data.received is set to true if the shipping
                // process is success
                verify(this.data = Subscribe(event); this.data.received)
                this.escrow.Pay(asset=asset, amount=amount, to=seller)
        }
        //called to finalize the contract when the timer expires def final() {
                if (!this.data.received)
                        this.escrow.Pay(asset=asset, amount=amount, to=buyer)
```

**Bank Guarantee**

A bank can issue a bank guarantee (BG) on behalf of a buyer to state that it will fulfill the payment obligation to the seller, in case if the buyer defaults the payment or does not have sufficient money when issuing the payment contract. The BG smart contract can be emulated as a form of composable escrow contract. In this contract, the buyer's credit consists of the expected value of deposit collaterals. The collateral can be monetary assets, virtual assets with monetary values, or another contract with expected values. The buyer can also deposit additional values while waiting for the payment condition. When the payment is actually needed, the monetary value in the escrow can be automatically transferred. In case if the payment is insufficient, the account can redeem additional values from deposit collaterals and transfer the value to the seller.

**Code Example**

```
contract BG(start=time(06/20/2018), end=time(08/20/2018)) {
        def init(contractors, guarantor, owner: address, bgFee: num,
                        amoCon: num, collCon: asset<USD>, amoGua: num,
                        collGua: asset<USD>, event: event<url>) {
        this.escrow = New Escrow(signers=[contractor, guarantor])
        this.escrow.Lock(asset=collCon, amount=amoCon, from=contractors)
        this.escrow.Lock(asset=collGua, amount=amoGua, from=guarantor)
        Account(contractors).Pay(asset=Asset(USD), amount=bgFee)
        this.data: Message
}
def main() {
        // assuming event.data.success is true if the contractors deliver
        // the project on time. The signal is triggered by a surveyor verify(this.data =
        Subscribe(event); this.data.success)
        final()
}
def final() {
        if (this.data.success) {
                this.escrow.Pay(asset=collCon, amount=amoCon, to=contractors)
                this.escrow.Pay(asset=collGua, amount=amoGua, to=guarantor)

        } else {
                // the project on time. The signal is triggered by a surveyor
                auditable(this.escrow.PayAllBalance(to=owner))
```

```
                }
        }
}
contract Contruction(start=time(06/20/2018), end=time(08/20/2018)) {
        def init(contractors, owner: address, bg: Contract, amount: num,
                wage: asset<USD>, event: event<url>) {
        if !audit(bg, escrow.PayAllBalance(to=owner))
                exit()
        this.escrow = New Escrow(signers=[contractor, guarantor])
        this.escrow.Lock(asset=wage, amount=amount, from=owner)
        this.data: MessageQ
        }
        def main() {
                // assuming event.data.success is true if the contractors deliver
                // the project on time. The signal is triggered by a surveyor
                verify(this.data = Subscribe(event); this.data.success)
                final()
        }
        def final() {
                if (this.data.success) {
                        this.escrow.Pay(asset=amount, amount=wage, to=contractors)
                } else {
                        wait(bg.isFinished()) this.escrow.PayAllBalance(to=owner)
                }
        }
}
```

**Loan**

Loans can be viewed as a two-stage contract where the borrower and the lender agree on the collateral and repayment terms. In the first stage, the borrower provides the expected value of deposit collaterals which can be redeemed such as virtual assets with monetary values, a payroll contract with future redeemable values, or a scheduled periodic payment contract from existing accounts. After the risk has been assessed and the loan has been approved, the loaner can then deposit the loan amount into the given accounts. The ongoing contract can then periodically redeem value from the provided collaterals.

**Code Example**

```
contract Loan(start=time(06/20/2018), end=time(08/20/2018)) {
    def init(lender, borrower: address,
            amount: num, collateral: asset<GOLD>,
            targetAmount: num, targetAsset: asset<USD>) {
        this.escrow = new Escrow(signers=[lender,borrower])
        this.escrow.Lock(asset=collateral, amount=amount, from=borrower)
        this.hasReceived, this.hasPaidBack : bool = false
    }
    def main() {
        verify(this.hasReceived = monitor(from=lender, to=borrower,
        amount=targetAmount,as set=targetAsset, source="stellar",
        memo="lendId"; this.hasReceived)
        )
        wait(time(08/19/2018))
        verify(this.hasPaidBack = monitor(from=borrower, to=lender,
            amount=targetAmount, asset=targetAsset, source="stellar",
            memo="paymentId"; this.hasPaidBack)
        )
    }
    def final() {
        if (this.hasReceived) {
            if (!this.hasPaidBack) {
                this.escrow.Pay(asset=collateral, amount=amount, to=lender)
            } else {
                this.escrow.Pay(asset=collateral, amount=amount, to=borrower)
            }
        }
    }
}
```

**Supplier Contract Chain**

The primary contractor first enters a procurement contract with a buyer. The primary contractor then enters into an agreement with a supplier by providing a limited reference to the original procurement contract or other assets. Upon execution, the EvryNet network can verify the availability of the fund and notify the supplier contract. The network will also keep the graph of a payment obligation. When the buyer issues payment to the primary contractor, EvryNet will examine the existing obligations graph and will have to transfer the initial amount to an escrow account. Once the supplier's payment is fulfilled, the remaining fund from the escrow will then be transferred to the primary contractor. These transactions are aggregated and recorded to the payment network after the settlement. In case of multiple installments, each contract needs to specify a synchronization point or "round" where transactions for a particular installment can be aggregated and settled before issuing a transfer order to the payment network.

**Code Example**

```
contract Parent(start=time(06/20/2018), end=time(06/25/2018)) {
        def init(employer, employee: address, amount: num, asset: asset<USD>) {
                this.escrow = new Escrow(signers=[employer,employee])
                this.escrow.Lock(asset=asset, amount=amount, from=employer)
        }
        def main() {
                wait(time(06/30/2018))
                this.escrow.Pay(asset=asset, amount=amount, to=employee)
        }
        def final() {}
}
contract Child(start=time(06/20/2018), end=time(06/30/2018)) {
        def init(employer, employee, creditCard: address,
                amount: num, asset: asset<USD>) {}
        def main() {
                verify(monitor(from=employer, to=employee, asset=asset,
                        source="stellar", memo="paymentId";))
                escrow.Pay(asset=asset, amount=amount, to=creditCard)
        }
        def final() {}
```

**Cross-chain Atomic Swap**

The EvryNet network supports atomic swap using hashed time-lock contracts. The swap operation consists of two parties: an initiator and another participant. The initiator first picks a secret which will be used to claim assets from both chains. The participant does not know the chosen secret until the initiator reveals it to claim an asset. There are four accounts involved in the process: initiator and participant accounts on both chains. The EvryNet network can generate a native script for any chain by using a broker-service e.g., Bitcoin-broker or Ethereum-broker. The time period used to lockup an asset in the multi-signature account on both chains can be calculated by a lockup service based on each user's preference and an acceptable transaction cost. Both initiator and participant must agree and initially sign on the generated native script for atomic swap on both chains. EvryNet network will then automatically monitor the chains and submit the transaction sequences to claim the assets for the initiator and participant on both chains.

**Code Example**

```
contract SwapBTCnXLM(start=time(06/20/2018), end=time(06/29/2018)) {
        def init(initiatorXLM, participantXLM: address, amountXLM: num,
                        initiatorBTC, participantBTC: bytes20, amountBTC: num,
                        secretHash: string, event: event<call>) {
                this.escrowXLM = new Escrow(signers=[initiatorXLM,
                        participantXLM])
                this.escrowBTC = new EscrowBTC(signers=[initiatorBTC,
                        participantBTC])
                this.escrowXLM.Lock(asset=Asset("XLM"), amount=amountXLM,
                        from=initiatorXLM)
                this.escrowBTC.Lock(amount=amountBTC, from=paricipantBTC)
                SetSecretHash([this.escrowBTC,this.escrowXLM])
                this.X: MessageCall
                this.hasClaim: bool = false
        }
        def main() {
                verify(this.X = Subscribe(event); hash(this.X) == secretHash)
                this.escrowBTC.PaywithX(amount=amount, to=initiatorBTC, this.X)
                this.escrowXLM.PaywithX(asset=Asset("XLM"), amount=amountXLM,
                        from=participantXLM, this.X)
                                this.hasClaim = true
        }
        def final() {
```

```
                if (!this.hasClaim) {
                        this.escrowBTC.Pay(amount=amount, to=participantBTC)
                        this.escrowXLM.Pay(asset=Asset(“XLM”), amount=amountXLM,
                                from=initiatorXLM)
                        }
        }
}
```