Overledger Network Community Treasury Design

Quant Network

March 2020

Abstract

This paper: describes aspects of the Overledger Network; discusses the community treasury's role in handling QNT payments between users and gateways using *layer 2 unidirectional payment channels*; and details how the community treasury enforces rules between the users and gateways, known as *payment contracts*, which dis-incentivises faulty behaviour using a game theoretic model. Using payment channels will allow QNT transfers between users and gateways flowing through the community treasury to be quick, significantly reduce their Ether fees and be publicly verifiable upon channel settlement. Additionally, the implementation of these payment contracts will be a breakthrough for the distributed ledger domain. With payment contracts, users not running a distributed ledger node can ask multiple gateways to perform a request where the responses will be compared for verification purposes. Therefore, not only do distributed ledger networks provide a method to establish trust between different parties running different nodes, but the Overledger Network will provide a method to establish trust between users not running a node and parties running nodes.

1 Background

In 2018 Quant Network released its white paper [1] introducing Overledger, a blockchain operating system, which allows for the design, deployment and execution of multi-ledger applications. Using Overledger you can connect or inter-operate current and future distributed ledgers (including blockchains), as well as legacy systems.

Additionally, in 2019 Quant Network released another white paper [2] introducing the Overledger Network, which will allow enterprise and distributed ledger community stakeholders to access and participate in a multi-ledger distributed network. The Overledger Network will be made up of gateways that link to various technologies, such as permissioned and permissionless distributed ledgers, as well as legacy systems. The Overledger Network is therefore set to solve the challenges of enterprises and distributed ledger communities operating in silos. Enterprises will be able to host their own secure dedicated gateways, resulting in access to new markets and new users across many different distributed

ledgers, without compromising their corporate security policies and regulation. Community members will also be able to host a gateway, and therefore take an active part in the first public distributed ledger interoperability project that does not introduce another blockchain.

The next milestone for the community portion of the Overledger Network roadmap is the design of the community treasury. The community treasury's role is to handle QNT payments flowing from users to the gateways in such a way as to disincentivise faulty behaviour from any user or gateway, and to do so in a manner where it can be held accountable to any observer.

This paper therefore describes the community treasury design and is structured as follows. Section 2 introduces the main aspects of the Overledger Network. Section 3 describes the main configuration choices of the stakeholders in the Overledger Network. Section 4 describes the payment contracts that are enforced by the community treasury. While section 5 concludes with some next steps.

2 Overledger Network Introduction

2.1 Overledger Data Standards

The Overledger Network relies on data standards so that all stakeholders in the network "speak a common language" and inconsistencies in responses can be detected. This data standard is known as the *Overledger Data Standard*, which is being introduced in a separate workstream.

Below contains formalisation that sets out the generic rules for this data standard. In summary, data object types describe the structure of objects whereas data object instances detail the instantiated versions of these data object types. Therefore there are many data object types (1), each contains zero or more attribute and data object type pairs (1,a). No two data object types are the same (1,b) but some data object types may contain a complete subset of pairs of other data object types (1,c), e.g. a generic UTXO transaction data object type will have a subset of pairs compared to what a Bitcoin transaction data object type has. Every data object type contains attributes listing its object type name and Overledger Data Standard version (1,d).

Additionally, there are many data object instances (2), again each contains zero or more attribute and data object instance pairs (2,a). Each data object instance is associated with a data object type if they share every attribute (2,b,i) and each attribute's data object instance is associated with the attribute's data object type (2,b,ii). No two data object instances are the same (2,c) but some data object instances may contain a complete subset of pairs of other data object instances (2,d), e.g. a generic UTXO transaction data object instance describing a particular transaction will have a subset of pairs compared to what a Bitcoin transaction data object instance describing the same transaction has.

1. Many *data objects types*, denoted $\mathcal{O} = \{O_1, O_2, ...\}$, where $null \in \mathcal{O}$. Objects in this set abide by the following rules:

(a) Each $O_x \in \mathcal{O}$ has zero or more (attribute a, object O) pairs denoted $O_x = \{\langle a_1, O_p\rangle, \langle a_2, O_q\rangle, ..., \langle a_m, O_t\rangle\}$ where $O_p, O_q, ..., O_t \in \mathcal{O}$

(b) For all $O_x, O_y \in \mathcal{O}$, then $O_x \neq O_y$

(c) There may be $O_x, O_y \in \mathcal{O}$ where $O_x \subset O_y$

(d) For all $O_x \in \mathcal{O}$ there $\exists\langle$"$objectName$", $O_r\rangle, \langle$"$objectVersion$", $O_s\rangle \in O_x$

2. Many *data object instances*, denoted $\mathcal{I} = \{I_1, I_2, ...\}$, where $null \in \mathcal{I}$. Objects in this set abide by the following rules:

(a) Each $I_x \in \mathcal{I}$ has zero or more (attribute a, instance I) pairs denoted $I_x = \{\langle a_1, I_p\rangle, \langle a_2, I_q\rangle, ..., \langle a_m, I_t\rangle\}$ where $I_p, I_q, ..., I_t \in \mathcal{I}$

(b) Each $I_x \in \mathcal{I}$ is associated to one data object type $O_x \in \mathcal{O}$, returned by $\texttt{Type}(I_x) = O_x$, iff for each $a_i \in \langle a_i, I_r\rangle \in I_x$ then:

i. $\exists\langle a_i, O_s\rangle \in O_x$ and

ii. $\texttt{Type}(I_r) = O_s$

(c) For all $I_x, I_y \in \mathcal{I}$, then $I_x \neq I_y$

(d) There may be $I_x, I_y \in \mathcal{I}$ where $I_x \subset I_y$

2.2 Overledger Network's Users and Gateways

As well as objects relating to the Overledger Data Standard, additional elements of the Overledger Network relevant to the community treasury design include:

- A set of *users* of the Overledger Network denoted $\mathcal{U} = \{u_1, u_2, ...\}$;
- A set of *functions* denoted $\mathcal{F} = \{f_1, f_2, ...\}$ that can be computed via the Overledger Network. Each function $f \in \mathcal{F}$ takes as input a data object instance of a specific data object type and returns as output another data object instance of another specific data object type. Functions are organised into four main sets[1]:
  - $F^{private} \subset \mathcal{F}$ contains all the functions that use private gateway data, e.g. `GET Licence`.
  - $F^{compare} \subset \mathcal{F}$ contains all the functions that do not use private data and the results from different gateways can be compared for verification purposes, e.g. `GET Transaction`.
  - $F^{check} \subset \mathcal{F}$ contains all the functions that do not use private data but a different function is more appropriate to use for verification purposes, e.g. `POST Transaction` should be verified not by calling the same function on another gateway but instead by calling `GET Transaction`. If $f \in F^{check}$ then the corresponding function f' to check it, is returned by $\texttt{Check}(f) = f' \in F^{compare}$.

[1] Note that multi-chain applications may contain many functions from the different subsets of $\mathcal{F}$.

– $F^{multi} \subset \mathcal{F} \setminus F^{private}$ contains functions that do not utilise private data of an Overledger gateway and do not utilitise the current state of a distributed ledger with probabilistic finality[2]. This is the set of functions that we can create multi-gateway payment contracts over, discussed in section 4;

- A set of *Overledger instances (also known as gateways)*, denoted $\mathcal{G} = \{g_1, g_2, ...\}$. The set of gateways not blacklisted by user $u \in \mathcal{U}$ is denoted $\mathcal{G}_u \subseteq \mathcal{G}$. The set of gateways implementing function f and not blacklisted by user u is denoted $G_{u,f} \subseteq \mathcal{G}_u$.

A user $u \in \mathcal{U}$ can make a request to a gateway $g \in \mathcal{G}$ to perform a function $f \in \mathcal{F}$. If a QNT payment is required for this function and gateway g resides on the community Overledger Network, then the flow of this QNT payment from the user u to the gateway g is managed by the community treasury multi-chain application, denoted $MAPP^C$.

To be a gateway $g \in \mathcal{G}$ in the community Overledger Network requires the owner of g to pay a minimum deposit $deposit^m$ (licence fee) and another variable deposit $deposit^{v,g}$ (serving as this gateway's stake in the network). The larger the $deposit^{v,g}$, the more requests that gateway g is authorised by the treasury $MAPP^C$ to handle in parallel. I.e. some portion of a gateway's deposit will be locked while it performs each request, incentivising gateways to perform all of the requests they accept. To explain further, if a gateway g accepts a function request and incorrectly performs it (e.g. the request is to get a transaction corresponding to a hash, but a different transaction is returned) then the gateway's variable deposit $deposit^{v,g}$ will be deducted (slashed). For this reason, the treasury $MAPP^C$ will only authorise a gateway g to perform a request if $deposit^{v,g}$ is large enough to cover all deductions for all of its incomplete requests.

To be a user $u \in \mathcal{U}$ of the network requires each u to pay a user deposit $deposit^u$ (that will be used to pay fees to gateways processing functions) and another deposit $deposit^d$ (to cover paying any penalties for raising incorrect disputes with the treasury). This dispute deposit is not expected to be large but is used to disincentivise a user from raising many unnecessary disputes.

2.3 Overledger Network's Community Treasury

For the treasury $MAPP^C$ to operate in a trustless manner, $MAPP^C$ will be using *layer 2 unidirectional payment channels*. Payment channels are required as QNT is currently placed on Ethereum, which has a low number of transactions per second and requires every on-chain transaction to pay fees in Ether. Each payment channel will:

- Hold a certain amount of QNT added by the creator of this channel;

[2]We resolve issues relating to probabilistic finality by requiring users to timestamp their requests, as discussed in section 3.3. Note that functions that utilitise the current state of a distributed ledger with probabilistic finality can be in either $F^{compare}$ or F^{check}.

- Expire after the declared time period, with all unclaimed QNT able to be reclaimed by the channel creator;
- Allow the receiver of this channel to claim x QNTs from this channel if:
  - The receiver can prove he had an off-chain transaction sent to him by the creator of this channel that totals x QNTs;
  - The channel has greater than x QNTs remaining; and
  - This off-chain transaction has not yet been used to claim QNT.

  Note that the channel will remain open after a claim by the receiver if the expiry time has not yet occurred.
- Allow the creator of the channel to increase it's QNT value and increase it's declared timeout period;

There will (at least initially) be two types of layer 2 unidirectional payment channels in use: (1) between users and the treasury $MAPP^C$; and (2) between the treasury $MAPP^C$ and gateways. A payment channel between a user u (sender) and $MAPP^C$ (receiver) is denoted $p^{u,C}$ and is initially funded with $deposit^u$. A payment channel between $MAPP^C$ (sender) and a gateway g (receiver) is denoted $p^{C,g}$ and is initially funded to the value of $limit^{C,g} \geq deposit^m$, where $limit^{C,g}$ is placed in the channel by the treasury (the true value of $limit^{C,g}$ can be dependant on many factors, e.g. licence, partnership, reputation, etc, and is set at the treasury's discretion).

These payment channels will operate as follows:

- There will be one on-chain transaction to create the channel with an opening QNT balance;
- Many off-chain transactions;
- Some on-chain transactions to increase the QNT balance of the channel and/or increase the time-out of the channel;
- Some on-chain transactions to redeem QNT balance from the channel;
- And one on-chain transaction to close the channel.

It is expected (and is in fact the purpose of payment channels) that the number of off-chain transactions for a payment channel will significantly outnumber the amount of on-chain payment channel transactions.

For a clear separation between fee payments and deductions (slashing) for bad behaviour, payment channels will strictly be used to process QNT fees flowing through the network whereas other smart contracts will be used to handle dispute payments. Therefore each user u's dispute deposit $deposit^d$ will be held not in channel $p^{u,C}$ but in another smart contract, denoted $sc^{u,C}$, where the treasury $MAPP^C$ can redeem QNT from the user under certain situations, e.g. the user raised an unnecessary dispute (discussed in section 4).

Likewise, each gateway g's $deposit^{v,g}$ will be held not in channel $p^{C,g}$ but in another smart contract, denoted $sc^{g,C}$, where $MAPP^C$ can redeem QNT from the gateway under certain situations, e.g. the gateway incorrectly computed a function (discussed in section 4). Both $sc^{u,C}$ and $sc^{g,C}$ will have the same time limit as $p^{u,C}$ and $p^{C,g}$ respectively, after which the locked funds can be withdrawn back to user u and gateway g respectively. There will be one final smart contract for disputes, one between the treasury and gateways, denoted $sc^{C,g}$, where a gateway g can redeem QNT from the treasury under certain situations, e.g. if the treasury did not pay the correct fee for the gateway processing a function (discussed in section 4). The dispute smart contract $sc^{C,g}$ does not have a time-limit but will operate a circuit-breaker (a.k.a. speed bump) withdrawal process. That is, if the treasury wants to withdraw funds from this contract, it will add an initial transaction stating its intentions (I want to withdraw this amount of QNT) and this withdrawal will only be able to complete after a certain number of blocks (the exact number is to be decided on and could be another variable that can be edited through a circuit breaker).

There are a few useful functions we can perform over payment channels and smart contracts for disputes:

- `TimeLimit`$(\{p, sc\}) \rightarrow \mathbb{R}$ returns the expiry time of the payment channel p (or deposit smart contract sc);
- `DepositedAmount`$(\{p, sc\}) \rightarrow \mathbb{R}$ returns the initial deposit amount and any additional deposits made by the sender of this payment channel p (or deposit smart contract sc) before its completion. Only the receiver can claim funds from p (or sc) before its time limit is reached;
- `PaidOnChainAmount`$(\{p, sc\}) \rightarrow \mathbb{R}$ returns the current total value settled on-chain from the sender of this payment channel p (or deposit smart contract sc) to the receiver.
- `PaidOffChainAmount`$(\{p, sc\}) \rightarrow \mathbb{R}$ returns the current total value settled off-chain from the sender of this payment channel p (or deposit smart contract sc) to the receiver, which can be claimed before its expiry time is reached.
- `LockedAmount`$(\{p, sc\}) \rightarrow \mathbb{R}$ returns the current total value locked on payment channel p (or deposit smart contract sc), while we await a function to be processed. Note that this value is locked off-chain at the treasury level;
- `Balance`$(\{p, sc\}) \rightarrow \mathbb{R}$ returns the current balance of this payment channel p (or deposit smart contract sc), equal to the following: `DepositedAmount`$(\{p, sc\})$ − `PaidOnChainAmount`$(\{p, sc\})$ − `PaidOffChainAmount`$(\{p, sc\})$ − `LockedAmount`$(\{p, sc\})$.

3 Overledger Network Configuration

In the Overledger Network, the three main categories of actors (gateways, treasury and users) have decisions to make on various parameters, as we describe below. Note that many of these variables may only need to be set once in the initial configuration and/or can be automatically updated according to some external variables (e.g. the number of gateways online, QNT to dollar exchange rate, etc).

3.1 Gateways

Each gateway $g \in \mathcal{G}$, must decide on the *variable deposit* to stake on the network. A gateway's total amount ever staked with treasury $MAPP^C$ will be returned by $\texttt{DepositedAmount}(sc^{g,C})$, whereas the gateway's current amount staked is returned by $\texttt{Balance}(sc^{g,C}) + \texttt{LockedAmount}(sc^{g,C})$.

Each gateway g should decide which functions it wants to support, denoted $F^g \subset \mathcal{F}$. The gateway's options here may be restricted by the gateway tier it is[3]. For each function $f \in F^g$ the gateway has to decide what *QNT fee* it wants to receive to perform the function, returned by $\texttt{Fee}(g, f)$.

Also any gateway has the right to *decide to process a function* $f \in \mathcal{F}$ at a particular time. I.e. any user can request any gateway to perform a function, but the gateway is not forced to accept this request. Only if a gateway accepts a request does it enter into a contract with the user regarding this function (these treasury enforced contracts are discussed in section 4). Reasons that the gateway may reject this request include: the gateway does not support the function (i.e. $f \notin F^g$); the gateway is currently operating at maximum capacity; the gateway is about to go (or is currently) offline; etc.

3.2 Treasury

The treasury $MAPP^C$, *can decide with which users and gateways to establish payment channels* (and therefore whose payments to oversee). Furthermore, for each $f \in \mathcal{F}$ that the treasury is overseeing, it needs to declare:

- A *dispute fee*, payable by a user that unnecessarily raises a dispute, returned by $\texttt{DisputeFee}(C, f)$ and chargeable per gateway evaluated.
- The *verification process* that the treasury will use to judge disputes, returned by $\texttt{VerificationProcess}(C, f)$. This may involve a manual process (e.g. check the object returned from the gateways against the data from the top 3 block explorers for that blockchain), and so includes a time cost (which is another motivation to have introduced $\texttt{DisputeFee}(C, f)$).
- A *commission fee* that the treasury can charge for a payment sent to gateway g for computing function f, returned by $\texttt{Commission}(C, g, f, \texttt{Fee}(g, f))$. Note that the commission fee can be zero or may be in some proportion

[3] See [2] for further information.

to the fee the gateway is charging (hence why $\texttt{Fee}(g, f)$ is passed in as the fourth variable).

- A *penalty fee* that a *gateway* g needs to pay if it computes the function incorrectly, denoted $\texttt{Penalty}(C, g, f, \texttt{Fee}(g, f))$. Note that again the penalty fee may be in some proportion to the fee the gateway is charging.
- A *penalty fee* that the *treasury* $MAPP^C$ needs to pay to a gateway if it does not pay it on time or pays incorrectly, denoted $\texttt{Penalty}(C, f, \texttt{Fee}(f, g))$.

For simplicity, the treasury can charge the same dispute fee for many (if not all) functions as well as use the same penalty and commission formulas for many (if not all) functions.

Note that as the treasury is responsible for connecting payments between users and gateways, it has a responsibility to *always be available* (or provide another payment route for users/gateways). This is unlike gateways that may go offline for one reason or another without being responsible for any other aspect of the Overledger Network.

Finally, a treasury needs to either offer its own *timestamping service*. Timestamping services will be essential to resolve disputes (as you will understand from section 4). This draws parallels between a treasury and a notary on the Corda network, hence why in the future there may be multiple Treasuries.

3.3 Users

When a user $u \in \mathcal{U}$ interacts with the Overledger Network it needs to have decided:

- What *function* $f \in \mathcal{F}$ to call.
- What *data object instance* $I \in \mathcal{I}$ to pass to the function. If $f \in F^{compare}$ and f interacts with a distributed ledger, then the following restriction is placed on I:
  - One time stamp per distributed ledger needs to be provided as well as the time stamp unit (taken from one of the options in the Overledger Data Standard). Gateways will then use this time stamp to understand from what ledger version (or block number), the user wants data fetching from. Note that if the distributed ledger in question uses probabilitistic finality and the sent timestamp is too close to the timestamp of the current state, a gateway may reject the request as the gateway will be awaiting a sufficient number of confirmations.
- What is the *maximum wait time for the function to return*, denoted T_f, *the maximum wait time for the function to be checked*, denoted $T_{\texttt{Check(f)}}$ and the *maximum time to raise a dispute*, denoted $T_{dispute}$. These wait times will be dependant on:
  - what type of function f is, i.e. is $f \in F^{compare}$ or is $f \in F^{check}$; and

– what is the expected time for this function to be completed with a high probability.

- The set of *desired numbers* $D = \{d^{compare}, d^{check}\}$ and the set of *minimum numbers* $M = \{m^{compare}, m^{check}\}$ *of gateways* to request a function $f \in F^{multi} \cap F^{compare}$ (respectively $f \in F^{multi} \cap F^{check}$) is performed on, where the results from each gateway will be verified against each other. Note that for a function $f \notin F^{multi}$ (i.e. a function that uses private data or the current state of a distributed ledger with probabilistic finality) then results from one gateway cannot be accurately verified using results from another gateway, as these functions are highly likely to produce different results for different gateways.

- What *metric* to use to choose which gateway(s) to ask to compute the function f, denoted $metric \in \{random, price, speed, distance, reputation, ...\}$.

- Regardless of the metric chosen an element of *randomness*, denoted *rand* where $0 < rand \leq 1$, needs to be introduced into the process so that gateways cannot definitively understand which other gateways may get asked to perform the same function.

- What *specific gateways to choose* for function $f \in \mathcal{F}$, returned by the function $\texttt{ChooseGateways}(G_{u,f}, D, M, metric, rand)$ in the form of a list of lists. If $f \notin F^{multi}$, there will only be one gateway g returned with its corresponding time out T_f, i.e. the returned object will be $[L^f = [\langle g, T_f \rangle]]$. If $f \in F^{multi} \cap F^{compare}$ there will be one list of gateways returned each with their own individual time outs, where each gateway will be asked to compute the same function f before their timeout, i.e. the returned object will be $[L^f = [\langle g_p, T_p \rangle, \langle g_q, T_q \rangle, ...]]$, where $|L^f| \geq m^{compare}$ and for each $\langle g_r, T_r \rangle \in L^f$ then $T_r \leq T_f$. Otherwise if $f \in F^{multi} \cap F^{check}$, then there will be two lists of gateways returned each with their individual time outs, where each gateway of the first list will be asked to compute the same function f before their timeout and each gateway of the second list will be asked to compute the same function $\texttt{Check}(f) \in F^{compare}$ only after completion of function f , i.e. the returned object will be $[L^f = [\langle g_p, T_p \rangle, \langle g_q, T_q \rangle, ...], L^{\texttt{Check}(f)} = [\langle g_r, T_r \rangle, \langle g_s, T_s \rangle, ...]]$, where: $|L^f| \geq m^{check}$; $|L^{\texttt{Check}(f)}| \geq m^{compare}$; for each $\langle g_r, T_r \rangle \in L^f$ then $T_r \leq T_f$; and for each $\langle g_s, T_s \rangle \in L^{\texttt{Check}(f)}$ then $T_s \leq T_{\texttt{Check}(f)}$.

Note again that many of these variables will/can be set automatically given initial configuration by the user.

4 Payment Contracts

When a user requests a gateway to perform a function $f \in \mathcal{F}$, we want to make sure that the function is correctly computed. If $f \notin F^{multi}$, then the user

must trust the gateway performing the function and the treasury organises the payments using what we call a *single gateway payment contract*.

Otherwise if $f \in F^{multi}$ (which will be the vast majority of functions that interact with a permissionless distributed ledger), a game theoretic approach can be used to establish trust[4]. To do so the treasury $MAPP^C$ allows a user to ask multiple gateways to compute the same function f or a corresponding function $\texttt{Check}(f)$. The different gateways are then essentially placed in competition, each correctly responding gateway g will be rewarded with $\texttt{Fee}(g, f)$ or $\texttt{Fee}(g, \texttt{Check}(f))$, each non-responding gateway will not be paid and each gateway g' incorrectly responding will be penalised by $\texttt{Penalty}(C, g', f, \texttt{Fee}(g', f))$ or $\texttt{Penalty}(C, g', \texttt{Check}(f), \texttt{Fee}(g', \texttt{Check}(f)))$. Gateways are further incentivised to be truthful as the penalty fee from gateways returning incorrect results will be moved to the gateways returning truthful results. This approach means that economically rationally gateways will act truthfully, whereas malicious gateways will be slashed and removed from the Overledger Network quickly. This game theoretic approach is enforced by the treasury and implemented via what we call a *multi-gateway payment contract*. These contracts rely on the ability to compare two data object instances (e.g. $I_x, I_y \in \mathcal{I}$) of the same data object type (e.g. $\texttt{Type}(I_x) == \texttt{Type}(I_y)$) to see if they are equivalent, returned by $\texttt{Equivalent}(I_x, I_y) \rightarrow \{true, false\}$. Equivalent in this sense does not mean the data object instances have to be exactly equal, as one instance could have more additional fields than the other, but both instances cannot have any conflicting attributes. These multi-gateway payment contracts also rely on the ability to confirm the output $f(I) = I_x$ matches the expected output of $\texttt{Check}(f)(I_x)$, returned by $\texttt{Match}(\texttt{Check}(f), I, I_x) \rightarrow \{true, false\}$. For example, if $\texttt{POST Transaction} \in F^{check}$ takes as input the data object instance I that describes the transaction and returns the transaction hash of the posted transaction as data object instance I_x, then when function $\texttt{Check}(\texttt{POST Transaction}) = \texttt{GET Transaction}$ uses the transaction hash I_x as input, it should return information on the same transaction as described in I.

Now the payment contracts will be described in more detail. In section 4.1. the stages of a multi-gateway payment contract will be described, whereas in section 4.2. the stages of the more simplified single gateway payment contract will be described.

4.1 Multi-Gateway Payment Contracts

A multi-gateway payment contract has a few stages that can be summarised as follows. Stage 1 sets the definition of the payment contract, including its initial participants and the details on what needs to be computed. Stage 2 checks that all participants can afford the contract and if so locks the required funds of the participants in their payment channels and dispute smart contracts. Stage 3 defines how the user processes the contract before its completion. Stage 4 describes how the user evaluates the results of the contract. Stage 5 is an optional

[4]The game theoretic approach takes inspiration from an academic paper [3].

stage detailing how the treasury evaluates a dispute raised by a user. Note that economically rational actors will want to avoid stage 5 as malicious actor's deposits will be deducted in this stage (possibly resulting in their complete removal from the Overledger Network). Stage 6 describes how the treasury settles the contract. Finally, stage 7 is another optional stage describing how gateways can raise disputes.

1. **The Payment Contract Definition**: The contract is to perform function $f \in \mathcal{F}$ using the input $I \in \mathcal{I}$ and (optionally) perform function $\texttt{Check}(f)$ using the input $f(I)$. The contract's participants are firstly the User $u \in \mathcal{U}$, and secondly (one or two lists of) gateways, denoted $gateways_{u,f,I}$, which are initially set equal to the response from function $\texttt{ChooseGateways}(G_{u,f}, D, M, metric, rand)$. A gateway $g \in L^f \in gateways_{u,f,I}$ is rewarded or punished as follows:

   - If the function is performed correctly by gateway g before time T_g, then $\texttt{Fee}(g, f)$ must be paid by u to gateway g.
   - If the function is performed correctly by g but later than T_g, no payment is given.
   - Otherwise if the function is performed incorrectly by g, the penalty $\texttt{Penalty}(C, g, f, \texttt{Fee}(f, g))$ is taken from g.

   A gateway $g \in L^{\texttt{Check}(f)} \in gateways_{u,f,I}$ is rewarded or punished in the same manner as above (except replace f for $\texttt{Check}(f)$).

   Any gateway has a right to refuse being included in $gateways_{u,f,I}$ and therefore refuse the contract stated above. Should a gateway refuse to be in list L^f (respectively $L^{\texttt{Check}(f)}$), the user will have to choose another gateway to replace it, if this list is now below the minimum number required, i.e. if $|L^f| < m^{\{compare,check\}}$, where $f \in F^{\{compare,check\}}$ (respectively if $|L^{\texttt{Check}(f)}| < m^{compare}$).

   If there is a dispute raised by u (because the objects returned by the gateways have not arrived, are not equivalent or do not match), the dispute will be resolved by the owner of $MAPP^C$ (Quant Network in this case).

   This section can be summarised by saying that multi-gateway payment contracts are proposed by a user u, agreed upon by gateways $gateways^{f,u,I}$ and enforced by the treasury $MAPP^C$.

2. **Locking the Contract**: For each gateway that has accepted the request, i.e. $g \in L^f \in gateways_{u,f,I}$, the $MAPP^C$ confirms that the following conditions hold. In summary, these conditions makes sure that all payment channels and dispute smart contracts will not timeout before the end of the dispute period and that they all have enough balance.

   (a) $\texttt{TimeLimit}(p^{u,C}) > T_{dispute}$ and $\texttt{Balance}(p^{u,C}) \geq \texttt{Fee}(g, f)$

   (b) $\texttt{TimeLimit}(sc^{u,C}) > T_{dispute}$ and $\texttt{Balance}(sc^{u,C}) \geq \texttt{DisputeFee}(C, f)$

(c) $\texttt{TimeLimit}(p^{C,g}) > T_{dispute}$ and $\texttt{Balance}(p^{C,g}) \geq \texttt{Fee}(g, f)$

(d) $\texttt{TimeLimit}(sc^{g,C}) > T_{dispute}$ and $\texttt{Balance}(sc^{g,C}) \geq \texttt{Penalty}(C, g, f, \texttt{Fee}(f, g))$

Whereas for each gateway $g \in L^{\texttt{Check}(f)} \in gateways_{u,f,I}$ the same conditional checks as above are made (except replace f for $\texttt{Check}(f)$). If any of the above conditions do not hold, then $MAPP^C$ will refuse to enforce the payment contract at this time, and either:

- The user u will have to top up his on-chain deposited fees/increase their timeouts (in the cases of (a) and (b)); or
- Gateway g will be removed from L^f (respectively $L^{\texttt{Check}(f)}$) and the user u may have to choose another gateway to add to the list if the list is now below the user-configured minimum size (in the case of (c) and (d)).

If the above conditions hold for a gateway $g \in L^f \in gateways_{u,f,I}$, then the treasury locks an amount on the payment channels and dispute smart contracts according to the following:

(e) $\texttt{LockedAmount}(p^{u,C}) = \texttt{LockedAmount}(p^{u,C}) + \texttt{Fee}(g, f)$

(f) $\texttt{LockedAmount}(sc^{u,C}) = \texttt{LockedAmount}(sc^{u,C}) + \texttt{DisputeFee}(C, f)$

(g) $\texttt{LockedAmount}(p^{C,g}) = \texttt{LockedAmount}(p^{C,g}) + \texttt{Fee}(g, f)$

(h) $\texttt{LockedAmount}(sc^{g,C}) = \texttt{LockedAmount}(sc^{g,C}) + \texttt{Penalty}(C, g, f, \texttt{Fee}(g, f))$

Again this locking occurs in the same manner for each gateway $g \in L^{\texttt{Check}(f)} \in gateways^{u,f,I}$ (except replace f for $\texttt{Check}(f)$).

3. **Contract Processing**: There are two sets of gateways related to this payment contract: (i) gateways that deliver a result before their timeout, denoted $G^D_{u,f,I}$, initially set equal to the empty set i.e. $G^D_{u,f,I} = \emptyset$; and (ii) gateways that do not deliver a result before their timeout, denoted $G^N_{u,f,I}$, again initially set equal to the empty set i.e. $G^N_{u,f,I} = \emptyset$. Both $G^D_{u,f,I}$ and $G^N_{u,f,I}$ get updated over time according to the following rules:

   - If the user receives $f(I) = I_g$ from a gateway $g \in L^f \in gateways_{u,f,I}$ before timeout T_g then:
     - $G^D_{u,f,I} = G^D_{u,f,I} \cup \{g\}$ (the gateway is added to the set of gateways who have delivered a result)
   - If the user does not receive a response from a gateway $g \in L^f \in gateways_{u,f,I}$ before timeout T_g then:
     - $G^N_{u,f,I} = G^N_{u,f,I} \cup \{g\}$ (the gateway is added to the set of gateways who have not delivered a result)
     - $G_{u_j} = G_{u_j} \setminus \{g\}$ (the gateway is blacklisted by the user, for a user-configured time)

As in other stages, if at some-point the number of gateways that could return a result is less than the minimal number for this function, another gateway g' can be asked to complete the function before the user selected maximum wait time for the function to return is reached, i.e. T_f. If gateway g' accepts the request, funds are locked in the same manner as described in the previous stage.

Note that the user and the gateways agree on the definitive time through the timestamping service provided by the treasury. This service could proceed as follows, each gateway sends a hash of the message it will send the user for timestamping to the treasury. The treasury returns the hash with its signature to the gateway. Then the gateway sends this signed hash along with the original message to the user, who can check that the hash of the returned object matches the timestamped hash. Additionally, the gateway must be able to provide through an end point the original message to authorised users (at least the treasury and the user) until at least the end of the dispute period, otherwise it may not be paid (for reasons discussed in stage (5,a)).

Finally recall that $gateways_{u,f,I}$ can contain two none empty lists. If this is the case, no gateway $g \in L^{\texttt{Check}(f)} \in gateways_{u,f,I}$ will be asked to process function $\texttt{Check}(f)$ before f has been fully evaluated by the user in the next stage. Only after a successful evaluation of f, will the user return to this stage for $\texttt{Check}(f)$ processing where each gateway $g \in L^{\texttt{Check}(f)} \in gateways_{u,f,I}$ is grouped into one of $G^D_{u,\texttt{Check}(f),I}$ or $G^N_{u,\texttt{Check}(f),I}$ according to the rules above (except replace f for $\texttt{Check}(f)$).

4. **Contract Evaluation**: Upon receiving the computation result from d gateways (i.e. $|G^D_{u,f,I}| = d$) or when the deadline T_f has passed, the user should do the following:

   (a) If $|G^D_{u,f,I}| \geq m$ and each pair of gateways $g_r, g_s \in G^D_{u,f,I}$ delivered the equivalent results (i.e. $\texttt{Equivalent}(I_{g_r}, I_{g_s}) = true$), then:

      i. if $L^{\texttt{Check}(f)} \notin gateways_{u,f,I}$ then the payment contract has successfully completed for the user and therefore it now skips to stage 6 to pay only the gateways who returned a result;

      ii. if $L^{\texttt{Check}(f)} \in gateways_{u,f,I}$ then the payment contract returns to stage 3 but this time to ask the gateways of $L^{\texttt{Check}(f)}$ to process function $\texttt{Check}(f)$ using as input the agreed result of $f(I)$.

   (b) If $|G^D_{u,f,I}| < m$ then the payment contract has a *time out issue* and the user u will raise a dispute with $MAPP^C$ to record this.

   (c) If there exists a pair of gateways $g_r, g_s \in G^D_{u,f,I}$ with non-equivalent results (i.e. $\texttt{Equivalent}(I_{g_r}, I_{g_s}) = false$), then the payment contract has a *non equivalence issue* and user u will raise a dispute with $MAPP^C$ to record and resolve this. Note that this type of dispute incurs a time cost, as most likely a human will perform the final dispute check.

If $L^{\texttt{Check}(f)} \in gateways_{u,f,I}$ and all gateways in $L^f \in gateways_{u,f,I}$ have returned equivalent results for $f(I)$, then the evaluation for $\texttt{Check}(f)$ will occur in the same manner as above for each gateway $g \in L^{\texttt{Check}(f)} \in gateways^{u,f,I}$ (except replace f for $\texttt{Check}(f)$ and if (a) occurs, the payment contract will skip immediately to stage 6, i.e. it will not repeat stage 3 again). There is also one additional stage of evaluation assuming that all gateways in $L^{\texttt{Check}(f)} \in gateways^{u,f,I}$ have returned equivalent results for $\texttt{Check}(f)(f(I))$:

(d) If the agreed results of $f(I)$ does not match the agreed results of $\texttt{Check}(f)(f(I))$, i.e. $\texttt{Match}(\texttt{Check}(f), I, f(I)) = false$, then the treasury has a *no match issue* and user u will raise a dispute with $MAPP^C$ to record and resolve this. Note that again this type of dispute incurs a time cost, as most likely a human will perform the final dispute check.

5. **Contract Evaluation Dispute (Optional)**: There are three categories of disputes raised by a user u that are handled by the treasury $MAPP^C$:

(a) *Time Out Issues*: The treasury checks that the gateways recorded by the user as returning a response ($G^D_{u,f,I}$) are the same as the gateways who sent a message hash in relation to this payment contract to the timestamping service before T_f. If there exists a gateway g that the treasury has recorded but the user has not (i.e. $g \notin G^D_{u,f,I}$), then the treasury will request the message from gateway g that corresponds to the timestamped hash. If the message from g is returned correctly (corresponding to the message hash), the message is returned to the user and this gateway is included in the delivered list (i.e. $G^D_{u,f,I} = G^D_{u,f,I} \cup \{g\}$). Note that gateways are incentivised to not time out for a user u, as if they do, they will be blacklisted by u and therefore will receive no more requests from u for the foreseeable future.

The timeout issue evaluation for $\texttt{Check}(f)$ occurs in the same manner as above (except replace f for $\texttt{Check}(f)$).

(b) *Non Equivalent Issues*: The treasury firstly computationally checks that there exists a pair of gateways $g_r, g_s \in G^D_{u,f,I}$ that have returned different results, i.e. where $\texttt{Equivalent}(f, I_{g_r}, I_{g_s}) = false$. If this does not hold, the user has raised an unnecessary dispute and will therefore be penalised as follows:

i. $\texttt{PaidOffChainAmount}(sc^{u,C}) = \texttt{PaidOffChainAmount}(sc^{u,C}) - (\texttt{DisputeFee}(f) \times |G^D_{u,f,I}|)$

Whereas if a non equivalence is found, the treasury will move to the verification stage, verifying function f against input I. For the verification stage, we introduce two more subsets of gateways, those that responded truthfully, denoted $G^T_{u,f,I}$, and those that responded falsely denoted $G^F_{u,f,I}$. Again both sets are initially set to empty, i.e. $G^T_{u,f,I} = G^F_{u,f,I} = \emptyset$.

How the treasury $MAPP^C$ performs this verification is dependant on the previously defined $\texttt{VerificationProcess}(f)$ function. Let I_C be the result of the successful completion of the verification check by $MAPP^C$ for function f and input I. Given I_C, a gateway $g \in G^D_{u,f,I}$ is judged as follows:

- if $\texttt{Equivalent}(f, I_g, I_C) = true$, then g was truthful and $G^T_{u,f,I} = G^T_{u,f,I} \cup \{g\}$
- if $\texttt{Equivalent}(f, I_g, I_C) = false$, then g operated in a faulty manner and $G^F_{u,f,I} = G^F_{u,f,I} \cup \{g\}$

Each gateway g that provided an incorrect response (i.e. $g \in G^F_{u,f,I}$), will be punished as follows:

ii. $\texttt{PaidOffChainAmount}(sc^{g,C}) = \texttt{PaidOffChainAmount}(sc^{g,C}) + \texttt{Penalty}(C, g, f, \texttt{Fee}(f_i, g))$

The non equivalent issue evaluation for $\texttt{Check}(f)$ occurs in the same manner as above (except replace f for $\texttt{Check}(f)$)

(c) *No Match Issues*: Resolving these issues follows a similar process to non equivalent issues. The treasury will firstly computationally check that $\texttt{Match}(\texttt{Check}(f), I, f(I)) = false$. If this does not hold, the user will be penalised in the same manner as part (b,i). If a no match issue is found, depending on the verification results, either the entire group of gateways in L^f or $L^{\texttt{Check}(f)}$ will be penalised as (b,ii) describes for f or $\texttt{Check}(f)$ respectively.

6. **Contract Settlement**: After time $T_{dispute}$, the treasury will settle the payment contract for each gateway g that correctly computed the function within its time limit (i.e. $g \in G^D_{u,f,I} \setminus G^F_{u,f,I}$) as follows. In summary, the user will pay the correct fee to each gateway (a). Each gateway will take its fee, minus any optional commission, as well as a portion of any penalties incurred by other gateways running the function (b). Finally all amounts locked on the payment channels and dispute smart contracts will be unlocked ((c), (d), (e), and (f)).

   (a) $\texttt{PaidOffChainAmount}(p^{u,C}) = \texttt{PaidOffChainAmount}(p^{u,C}) + \texttt{Fee}(g, f)$

   (b) $\texttt{PaidOffChainAmount}(p^{C,g}) = \texttt{PaidOffChainAmount}(p^{C,g}) + \texttt{Fee}(g, f) - \texttt{Commission}(C, g, f, \texttt{Fee}(g, f)) + \frac{\sum_{g' \in G^F_{u,f,I}} \texttt{Penalty}(C, g', f, \texttt{Fee}(g', f))}{|G^T_{u,f,I}|)}$

   (c) $\texttt{LockedAmount}(p^{u,C}) = \texttt{LockedAmount}(p^{u,C}) - \texttt{Fee}(g, f)$

   (d) $\texttt{LockedAmount}(sc^{u,C}) = \texttt{LockedAmount}(sc^{u,C}) - \texttt{DisputeFee}(C, f)$

   (e) $\texttt{LockedAmount}(p^{C,g}) = \texttt{LockedAmount}(p^{C,g}) - \texttt{Fee}(g, f)$

   (f) $\texttt{LockedAmount}(sc^{g,C}) = \texttt{LockedAmount}(sc^{g,C}) - \texttt{Penalty}(C, g, f, \texttt{Fee}(g, f))$

   The treasury will also unlock the payment channels and deposit smart contracts for each gateway g' that either timed out or was faulty (i.e.

$g' \in G^N_{u,f,I} \cup G^F_{u,f,I}$) by performing only steps (c) to (f) above (replace g for g'). Note that any penalty fees will have already been decremented in stage 5.

The contract settlement occurs for function $\texttt{Check}(f)$ in the same manner as above (except replace f for $\texttt{Check}(f)$).

Note that the treasury and the gateway can claim any paid off-chain amount at any time in the future (before the relevant time limit) by posting an Ethereum transaction.

7. **Contract Settlement Dispute (Optional)**: The final stage allows for a gateway g to dispute a payment received from the treasury. The full definition of this stage requires further detail on what the off-chain messages sent between the user, treasury and gateway contain, and when these messages are sent. This information is held for another document. But to summarise, to raise a contract settlement dispute, a gateway sends a transaction to the dispute smart contract $sc^{C,g}$ with proof of underpayment and then the treasury $MAPP^C$ has until a certain time in the future to post the correct payment transaction on-chain or some of $MAPP^C$'s funds in $sc^{C,g}$ will get deducted and released to the gateway g.

4.2 Single Gateway Payment Contracts

Single gateway payment contracts are a simplified version of multi-gateway payment contracts and are summarised as follows:

1. **The Payment Contract Definition**: The contract is to perform function $f \in \mathcal{F}$ using the input $I \in \mathcal{I}$. The contract's participants are firstly the User $u \in \mathcal{U}$ and secondly a single gateway $g \in \mathcal{G}$ who has until T_f to perform the function. This gateway has the same reward or punishment as detailed in stage 1 of the multi-gateway payment contract section.

   This section can be summarised by saying that single gateway payment contracts are proposed by a user u, agreed upon by a gateway g and enforced by the treasury $MAPP^C$.

2. **Locking the Contract**: The treasury $MAPP^C$ confirms that conditions (a) and (c) described in stage 2 of the multi-gateway payment contract hold. Condition (b) is not required as the treasury does not handle non equivalent or no match issues for single gateway payment contracts and so there is no concern for users raising unnecessary disputes on these issues. Condition (d) is not checked as the treasury will not verify the data returned to the user from a single gateway. For these reasons, only sections (e) and (g) from stage 2 of the multi-gateway payment contract are locked.

3. **Contract Processing**: In this stage the user simply waits for the gateway to return a timestamped response.

4. **Contract Evaluation**: Upon receiving the computation result either:
    - The user has received a response from the gateway, in which case the payment contract skips to stage 6 to settle the payment.
    - Or the gateway has timed out, in which case the user raises a dispute to the treasury.
5. **Contract Evaluation Dispute (Optional)**: The treasury can only settle time out disputes for single gateway payment contracts, as described in section (a) of stage 5 of the multi-gateway payment contract.
6. **Contract Settlement**: The treasury will settle the contract using similar formulas to the ones declared in stage 6 of the multi-gateway payment contract. The only differences are: (b) cannot include any redistribution of penalty fees, as no other gateway is being asked to perform the function; also (d) and (f) are not valid as these values were never locked for single gateway payment contracts.
7. **Contract Settlement Dispute (Optional)**: Finally, gateways can still dispute the payment received from the treasury in the same manner as described in stage 7 of the multi-gateway payment contract.

5 Conclusion

To conclude, we have discussed the underpinnings of the Overledger Network and the community treasury's role within it. We have detailed how the community treasury will use payment contracts to incentivise truthful behaviour by gateways in the network and slash bad actors. The next steps of the community treasury design will be: to create further documentation describing the messages passed between the user, gateways and treasury; to detail how the on-chain dispute resolution smart contracts operate; and afterwards to create a publicly available prototype.

References

[1] Quant Network. Quant Overledger Whitepaper. [see: www.quant.network], 2018.

[2] Quant Network. Overledger Network: The network of networks for a hyper-connected world. [see: www.overledger.network], 2019.

[3] Changyu Dong, Yilei Wang, Amjad Aldweesh, Patrick McCorry, and Aad van Moorsel. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, Dallas, TX, USA*, pages 211–227. ACM, 2017.