# DORA: Distributed Oracle Agreement

Supra Research

January 26, 2023

**Abstract**

Oracle networks feeding off-chain information to a blockchain are required to solve a distributed agreement problem since these networks receive information from multiple sources and at different times. We make a key observation that in most cases, values obtained by oracle network nodes from multiple information sources are in close proximity. We define a notion of agreement distance and leverage the availability of a state machine replication (SMR) service to solve this distributed agreement problem with an honest simple majority of nodes instead of the conventional requirement of an honest super majority of nodes. Values from multiple nodes being in close proximity forming a coherent cluster, is one of the keys to its efficiency. Our asynchronous protocol also embeds a fallback mechanism if the coherent cluster formation fails. Through simulations using real-world exchange data from seven prominent exchanges, we show that even for very small agreement distance values, the protocol would be able to form coherent clusters and therefore, can safely tolerate up to $\frac{1}{2}$ of Byzantine nodes. We also show that, for a small statistical error, it is possible to choose the size of the oracle network to be significantly smaller than the entire system that tolerates up to a $\frac{1}{3}$ fraction of Byzantine failures. This allows the oracle network to operate much more efficiently and horizontally scale much further.

## 1 Introduction

Connecting existing Web 2.0 data sources to blockchains is crucial for next-generation blockchain applications such as decentralized finance (DeFi). An oracle network [4, 28] consisting of a set of interconnected and independent nodes aims to address this issue by allowing smart contracts to function over inputs obtained from existing Web 2.0 data, real-world sensors, and computation interfaces.

Performing this information exchange securely, however, is not a trivial matter. First, only a few data sources may be available to pick from, some of them may crash (due to a *Denial of Service* (DOS) attack or system failures), or even send incorrect information (due to a system compromise or some economic incentives) [10, 16]. Second, as most of the data sources today do not offer data in a signed form, an oracle network also becomes vulnerable due to the compromise of a subset of oracle network nodes, a subset of data sources, or due to the compromise of a combination of the two. Third, an adversary may go after the availability of the system (and at times safety) by malevolently slowing down the protocols. We address these issues and propose a robust and scalable distributed solution for solving the oracle problem. Our approach can withstand extreme adversarial settings. We make some real-world synchrony and input-data distribution observations, and introduce oracle execution sharding. This makes our solution scale extremely well as the number of oracle services and the size of our oracle network grows.

One of the key objectives of an oracle service is to take a piece of off-chain information and bring it to the on-chain world. Therefore, any such service must have three components, (i) sources of information, which we shall refer to as data sources, (ii) a network of nodes, and (iii) a target component in an on-chain environment (*i.e.*, a smart contract). To maintain fault tolerance capabilities as well as the decentralized nature of the service, it is necessary to have multiple data sources in addition to having a network of multiple nodes. The oracle agreement problem focuses on producing a unique value that is representative of the values emanating from the honest data sources that are feeding information to the oracle network. We need a protocol for ensuring that all the honest oracle nodes have the same output that is representative of all the honest data sources. Notice that this is non-trivial as all the honest nodes may still have different outputs since they may be listening to different sets of data sources at slightly different times. We call this problem a *Distributed Oracle Agreement* (DORA) problem. DORA shares the same termination and agreement properties with the

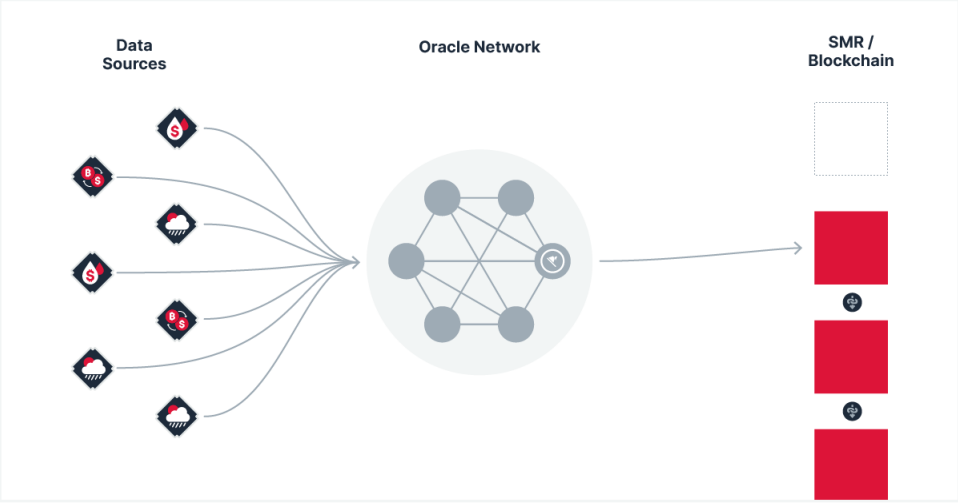



Figure 1: ABC protocol: Network of oracle nodes obtaining data from data sources, running a Byzantine agreement protocol and then broadcasting the agreed upon value to the world via SMR/Blockchain

well-studied *Byzantine agreement* (BA) problem [22, 24]. However, the crucial validity property for DORA is significantly more generic. BA demands that the output be the same as an honest node's input if all the honest nodes have the same input. DORA is a generalization of this, where the output will be a value within a range (convex hull) defined by the minimum and maximum honest inputs. As DORA generalizes the BA problem, it requires the standard 67% honest majority among the participating nodes. As the system scales and the number of participating nodes increases, solutions to DORA may not scale, especially when we collect many different kinds of variables. For a full-fledged blockchain system, we expect the number of variables for which representative values are to be computed to be in the order of several hundred.

Let us first understand how a traditional oracle protocol, denoted as ABC protocol, works. An oracle network consisting of multiple nodes obtains different values from multiple sources of information as shown in Figure 1 and agrees on a single report/output. As we discuss in Section 3.2, this problem is closely related to the standard Byzantine agreement problem [30]. In presence of Byzantine nodes and non-synchronous communication links, the oracle network has to have at least $3f + 1$ nodes, where $f$ is the upper-bound on the number of nodes that can turn Byzantine during the agreement protocol [19]. These oracle nodes run a Byzantine agreement protocol in order to agree on a subset of $2f + 1$ values. In ABC, one of the nodes is designated as a *leader* which would send this agreed-upon subset to the Blockchain. If the leader happens to be Byzantine, after a certain timeout period, the ABC protocol initiates a leader change.

A crucial point to note is that in ABC, the SMR/Blockchain is used merely as a means to broadcast/publish the agreed-upon set to the world. Even without the SMR/Blockchain, ABC protocol ensures that all the honest nodes of the oracle network agree upon the same subset of $2f + 1$ values. In a network with $f$ Byzantine nodes, $2f + 1$ values are required to ensure that statistical aggregation via computing the median does not deviate *too much* from the values received from honest nodes. Essentially, $2f + 1$ values ensure that the median is upper-bounded and lower-bounded by values from the honest nodes.

Blockchain ensures that the messages on the chain have a total order. Further, it also ensures that any entity observing the state of a Blockchain would witness the same total order. Not utilizing the ability of a Blockchain to act as an ordering service is a missed opportunity by oracle networks in general.

## 1.1 Our Approach

We leverage the ordering capabilities of the Blockchain by presenting a protocol that both sends and receives information from the Blockchain to accomplish the goal of publishing a value on the Blockchain that is representative of the values from all the honest data sources. Towards computing a representative value, we redefine the notion of agreement. We say that two nodes *agree* with each other if the values that they obtained from data sources are within a pre-defined parameter called *agreement distance*. We say that a

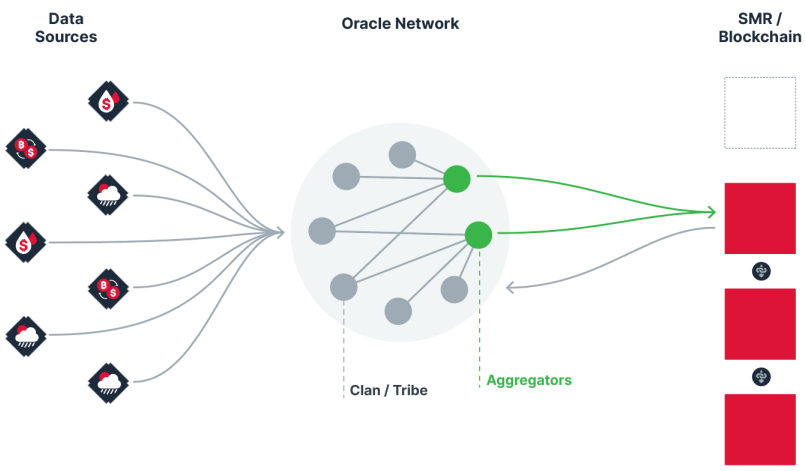



Figure 2: The oracle nodes collect information from various data sources. The oracle nodes exchange information with the aggregators to vote for the proposal of a coherent cluster of size $f+1$. The aggregators post these clusters to Blockchain and all oracle nodes consider the first cluster to be authoritative.

set of values form a *coherent cluster* if all the values in that set are at most agreement distance away from one another. The oracle network now merely needs to agree on a coherent cluster of size $f+1$. Since there would be at least one honest value in any such coherent cluster, any statistical aggregator such as the mean or the median would be at most agreement distance away from an honest value, thus ensuring that the final agreed upon value does not deviate *too much* from the honest values.

The way our protocol achieves agreement on a coherent cluster of size $f+1$ is that the first such cluster posted on the Blockchain would be considered authoritative for a given round of consensus.

Similar to ABC protocol, we also have designated nodes that we call *aggregators*. These aggregators would gather $f+1$ signatures on a proposed set of $f+1$ values provided by the nodes and then post it on the SMR/Blockchain. To circumvent the problem of having a Byzantine aggregator, we sample a set of aggregators, henceforth called a family of aggregators, from the entire oracle network such that there is at least one honest aggregator. This helps us avoid having to initiate any aggregator change protocol. So now, we have multiple aggregators posting sets of size $f+1$ to the Blockchain, but only the first one would be accepted as authoritative by the oracle network. Note that all such coherent clusters of size $f+1$ contain values that are at most agreement distance away from some honest value.

With the redefined notion of agreement and utilizing the ordering capability of the Blockchain, we reduce the number of oracle nodes required to $2f+1$ instead of the usual requirement of $3f+1$. Figure 2 shows how information flows across various components of our oracle protocol. Note that now the oracle nodes only have to communicate back and forth with the family of aggregators. All the oracle nodes and aggregators observe the total order in which information appears on the SMR/Blockchain. Aggregators are the only nodes that sends information to the Blockchain.

Under unusual circumstances, such as when none of the aggregators is able to form a coherent cluster, our protocol switches to a fallback mechanism. In this fallback mechanism, the requirement of total nodes again increases to $3f+1$ out of which the aggregators wait for $2f+1$ nodes to provide a value and compute a median. In this fallback mechanism, the arithmetic mean can not be used anymore since the values introduced by the Byzantine nodes in this set of size $2f+1$ could be unbounded. Therefore, our fallback mechanism, along the same lines as ABC, computes a median from the agreed upon set of size $2f+1$.

Our idea of the reduced requirement for the number of nodes hinges upon the honest nodes producing values in close proximity most of the time. To prove that this assumption is practical and well-founded, we take BTC price information from 7 exchanges that include FTX, and run a simulation of our protocol on this data covering a 30 day period that includes the turbulent period of FTX collapse [1]. We run simulations where the round of agreement happens every 30 and 60 seconds. These simulations corroborate our close proximity of honest values assumption. We observe that for agreement distances as low as \$25 and \$53,

when the average BTC price is around $19605, a coherent cluster can be formed in 93% and 99% rounds of agreement respectively.

To further scale our solution, we introduce execution sharding by sampling multiple sub-committees from the $3f+1$ nodes available in the entire system. The size and the number of such sub-committees are chosen such that each sub-committee has an honest simple majority with a very high probability. Since under normal circumstances we only need an honest simple majority, we equally divide the responsibilities to track multiple variables and bring their price information on-chain equally among these sub-committees. Since a sub-committee can now only provide a probabilistic safety guarantee, we analyze it in Section 4.2 and establish that with a few hundred nodes, the safety guarantee holds with a very high probability.

The rest of the paper is organized as follows. Section 2 introduces notations used in the paper. We describe our protocol in Section 3. We provide details of empirical analysis and safety analysis for probabilistic safety in Section 4. We provide details on some extensions in Section 5. Section 6 discusses some relevant aspects of the architecture and the design of our oracle.

# 2 Preliminaries

In this section, we introduce some preliminaries and notations to which we will refer for the remainder of this paper.

## 2.1 Oracle Network

Let $|S|$ denote the cardinality of a set $S$. We shall use $\mathcal{Q}_{0.5}(X)$ to denote the median of a set $X$ of values. Our oracle network consists of a set $N_t$ of $|N_t|$ nodes, which we also call a tribe. Among these, at most $f_t < \frac{|N_t|}{3}$ nodes may become compromised, that is to turn Byzantine, thus, deviating from an agreed-upon protocol and behaving in an arbitrary fashion. A node is *honest* if it is never Byzantine. We assume a static adversary that corrupts its nodes before the protocol begins.

From these $N_t$ nodes, towards developing scalable solution, we also uniformly randomly draw sub-committees, henceforth called clans. Each clan $N_c$ of size $|N_c|$ are drawn such that at most $f_c < \frac{|N_c|}{2}$ nodes are Byzantine in a clan. The clan nodes are sampled uniformly at random from the tribe, and we ensure that the probability of any clan having more than $\lfloor \frac{|N_c|}{2} \rfloor$ Byzantine nodes is negligible. Probability analysis for such sampling is provided in Section 4.2.

All oracle nodes are connected by pair-wise authenticated point-to-point links. We assume this communication infrastructure to be asynchronous such that the (network) adversary can arbitrarily delay and reorder messages between two honest parties. As typical for all asynchronous systems, for the system's liveness properties, we assume that the adversary cannot drop messages between two honest parties.

We initialize the protocol with a unique identifier to prevent replay attacks across concurrent protocol instances, but do not explicitly mention this in the text.

A signed message $m$ from a node $p_i$ are denoted by $m(\cdot)_i$. Similar to the most recent SMR/blockchain designs, we assume a $(n, n-f)$ threshold BLS [5,7] signature setup. We denote an $n-f$ threshold signature on the message $m$ as a quorum certificate $\mathcal{QC}m$. While the use of threshold signatures offers a simple abstraction and can be verified on Ethereum, we can also employ a multi-signature (multisig) setup as allowed by the employed blockchain.

## 2.2 Oracle Data Sources

A data source is responsible for providing the correct value of a variable $\tau$, which could be, say, the price of Bitcoin in US Dollars. Let $DS$ denote a set of data sources and $BDS \subset DS$ denote the subset of these data sources which could be Byzantine. We say that a data source is Byzantine if : (i) it lies about the value of the variable, or (ii) if it is non-responsive. Otherwise, we will consider the data source to be honest. We assume that $|BDS| \leq f_d$, where $f_d$ is the upper bound on the number of Byzantine data sources.

The goal of the tribe is to reach a consensus in a distributed fashion about a *representative value*, denoted as $\mathcal{S}$, of a particular $\tau$. The notion of a representative value depends on the underlying $\tau$. For example, we

can say that the representative value of a particular stock could be considered to be a mean of the values of the stock at various stock exchanges.

In this paper, for now, we assume that for the variable $\tau$ of interest, the arithmetic mean $\mu$ of values of $\tau$ from various data sources is the representative value.

An observation of a node $p_i$ from a data-source $ds_j$ of a variable $\tau$ is denoted as $o(p_i, ds_j, \tau)$. We say that an observation $o(p_i, ds_j, \tau)$ is an honest observation if both $p_i$ and $ds_j$ are honest. In an ideal world without any Byzantine nodes or Byzantine data sources, we would like the protocol to have the following property:

**Property 1** (Ideal Representative Value)**.** $\mathcal{S} = \frac{\sum_{ds_j \in DS} o(p_i, ds_j, \tau)}{|DS|}$ *where $p_i$ refers to any one of the honest nodes.*

When the context is clear, we will just use $o$ to denote an observation. Let $O$ denote the set of all observations. We will use $O_{p_i}$ to denote the observations made by node $p_i$. $\mathcal{H}(O)$ and $\mathcal{B}(O)$ denote the set of honest observations and Byzantine observations respectively. Let $\mathcal{H}_{min}(O_{p_i}) = \min_{\mathcal{H}(O_{p_i})}$ and $\mathcal{H}_{max}(O_{p_i}) = \max_{\mathcal{H}(O_{p_i})}$ indicate the minimum and maximum values from a given set of honest observations $\mathcal{H}(O_{p_i})$. We will just use $\mathcal{H}_{min}$ and $\mathcal{H}_{max}$ to refer to the minimum and the maximum values amongst all honest observations $\mathcal{H}(O)$.

We say that two observations $o_1$ and $o_2$ agree with each other if $\|o_1 - o_2\|_1 \leq \mathfrak{d}$. That is if the $L_1$ distance between them is at most $\mathfrak{d}$, where $\mathfrak{d}$ is a pre-defined parameter known as *agreement distance*. A set of observations $CC \subseteq O$ is said to form a *coherent cluster* if $\forall_{o_1, o_2 \in CC} : \|o_1 - o_2\|_1 \leq \mathfrak{d}$.

We will use the terms *majority* and *super majority* to denote that some entity has a quantity strictly greater than $\frac{1}{2}$ and $\frac{2}{3}$ fraction of the total population respectively. For example, an honest majority within a set of nodes would indicate that more than half of the nodes are honest. An honest super-majority would indicate the fraction of honest nodes to be strictly greater than $\frac{2}{3}$ of all the nodes within the set.

Let $\mathcal{S}_r$ denote the value for which the oracle network achieved a consensus for it to be considered as the representative value for a round $r$. We will use $\mathcal{S}_{r-1}$ to denote the value emitted by the oracle network in the previous round.

#### 2.2.1 Expected Representative Value with Byzantine Actors

We consider two kinds of bad actors in the system: (i) Byzantine data sources, and (ii) Byzantine nodes. We assume that Byzantine nodes and Byzantine data sources can collude in order to (i) prevent the oracle network from reaching a consensus in a given round, or (ii) prevent the oracle network from achieving Property 1.

In the presence of such Byzantine actors, even if a single dishonest observation gets considered for computation of the mean, the Byzantine nodes can be successful in forcing $\mathcal{S}$ to deviate arbitrarily from the true representative value equivalent to a mean[1]. Moreover, for the setting with $|N_t| \geq 3f_t + 1$ nodes, a rushing adversary can suggest its input only after observing the honest parties' inputs. Therefore, we can only aim for the following weaker property:

**Property 2** (Honest Bounded Value)**.** $H_{min} \leq \mathcal{S} \leq H_{max}$

As discussed in the literature [25], the agreed value is in the convex hull of the non-faulty nodes' inputs.

### 2.3 Primitive: State Machine Replication

In the form of state machine replication (SMR) [33] or blockchain, we employ a key distributed service for our system. An SMR service employs a set of replicas/nodes collectively running a deterministic service that implements an abstraction of a single, honest server, even when a subset of the servers turns Byzantine. In particular, an SMR protocol orders messages/transactions $tx$ from clients (in our case aggregators) into a totally ordered log that continues to grow. We expect the SMR service to provide public verifiability. Namely, there is a predefined Boolean function $\mathsf{Verify}$; a replica or a client outputs a log of transactions log $= [tx_0, tx_1, \ldots, tx_j]$ if and only if there is a publicly verifiable proof $\pi$ such that $\mathsf{Verify}(log, \pi) = 1$.

Formally, an SMR protocol [26] then provides the following safety and liveness:

[1] Byzantine oracle nodes are problematic particularly as data sources do not sign their inputs.

**Property 3** (Safety). *If* $[tx_0, tx_1, \ldots, tx_j]$ *and* $[tx'_0, tx'_1, \ldots, tx'_{j'}]$ *are output by two honest replicas or clients, then* $tx_i = tx'_i$ *for all* $i \leq \min(j, j')$.

**Property 4** (Liveness). *If a transaction tx is input to at least one honest replica, then every honest replica eventually outputs a log containing tx.*

We assume that there is an SMR/blockchain service running in the background that an oracle service can employ. Oracle network nodes are assumed to employ a simple put/get interface to the SMR service. They employ $\mathsf{post}_{\mathsf{SMR}}(\cdot)$ to post some (threshold) signed information (or transaction) on the SMR chain. Upon collecting and processing ordered transactions on SMR, the nodes employ "**Upon** witnessing" event handling to process the relevant messages. As communication links between oracle nodes and SMR service nodes are expected to be asynchronous, this put/get interface is expected to function completely asynchronously and provide guarantees that can be observed as an interpretation of SMR safety and liveness: (i) senders' messages appear on the blockchain eventually; (ii) different receivers observe messages at different points in time; (iii) however, all the nodes eventually observe messages in the exact same total order.

Unlike a few recent efforts (such as [14,23]) that treat blockchain as a (bounded-synchronous) broadcast channel with bounded-message delivery time, we find our weaker asynchronous primitive to be a better representative of modern blockchains. In other words, the broadcast channel assumption puts a strict time bound on message delivery via the blockchain. It is difficult though to offer such a precise bounded time mechanism for an honest transaction to appear on a blockchain in the presence of network asynchrony over the Internet, transaction reordering, and frontrunning.

# 3 The Supra Oracle Protocol

## 3.1 Data Feed Collection

The first step of an oracle service involves oracle nodes connecting with the data sources. As we assume that there are multiple data sources, some of them can be compromised and most of them do not sign their data feeds. Our key goal is to ensure that the honest (oracle) node's input to aggregators is representative of the honest data sources.

Towards ensuring the correctness of the honest nodes' inputs, we expect them to retrieve feeds from multiple data sources such that the median of the received values is representative of the honest values. In another words, it is inside the $[\mathcal{H}_{min}, \mathcal{H}_{max}]$ range of the honest data sources.

In this direction, we make a key synchrony assumption about communication links between data sources and oracle nodes. Unlike communication links between oracles nodes, we assume that links connecting data sources and oracle nodes to be bounded-synchronous such that if a node does not receive any value from the data source over the web API/socket in a time-bound $T_{ds}$, the node can assume that the source has become faulty/Byzantine.

In this bounded-synchronous communication setting, data feed collection works as shown in Algorithm 1.

1. We expect that up to $f_d$ data sources may become Byzantine.

   Therefore, out of abundant caution, we mandate that every oracle node connects to $2f_d+1$ data sources [2].

2. Every node sends a request to their assigned set of data sources $ADS$ and then starts the timer $T_{ds}$. (Lines 1 to 4)

3. Whenever a value $v$ is received from a data source $ds$, the node stores in $obs[ds]$ . (Line 7)

4. Upon *timeout* of $T_{ds}$ gather all the values received so far in $Obs$. (Line 10)

**Theorem 1** (Validity). *At the end of Algorithm 1,* $\mathcal{H}_{min}(Obs) \leq \mathcal{Q}_{0.5}(Obs) \leq \mathcal{H}_{max}(Obs)$.

[2] Our bounded-synchrony assumption for the source-to-node link is for simplicity. If these links behave more asynchronously, we can easily make the node contact $3f_d + 1$ data sources and wait to hear back from at least $2f_d + 1$ data sources to ensure that the honest nodes select a representative value.

*Proof.* We will prove the theorem by contradiction. Without the loss of generality, let us assume that $\mathcal{Q}_{0.5}(Obs) > \mathcal{H}_{max}(Obs)$. The total number of all the honest values received by a node is at least $f_d+1$ and the total number of all the Byzantine data sources is at most $f_d$ ($\mathcal{B}(Obs) \leq f_d$). Since all the honest data sources would report the correct value within $T_{ds}$, we would have $f_d + 1 \leq |Obs|$. Since $\mathcal{Q}_{0.5}(Obs) > \mathcal{H}_{max}(Obs)$ and $|\mathcal{H}(Obs)| \geq f_d + 1$, it must be the case that $|\mathcal{B}(Obs)| \geq (f_d + 1)$, a contradiction. One can similarly argue that $\mathcal{Q}_{0.5}(Obs) < \mathcal{H}_{min}(Obs)$ is not possible. □

**Algorithm 1** GetDataFeed($ADS$)

**Require:** $(ADS \subseteq DS) \wedge (|ADS| = 2f_d + 1)$

**Upon** init **do**
  $\forall_{ds \in ADS} obs[ds] \leftarrow \perp$
  send the request to all $ds \in ADS$
  start the timer $T_{ds}$

**Upon** receiving value $v$ from $ds$ **and** $T_{ds} > 0$ **do**
  $obs[ds] \leftarrow v$

**Upon** *timeout* of $T_{ds}$ **do**
  **return** $Obs = \{obs[ds] | obs[ds] \neq \perp\}$

## 3.2 Distributed Oracle Problem Definition

Once we ensure that every honest oracle node has produced a correct/representative value as its input, the oracle problem becomes a bit simpler. Since honest observations may still differ, therefore, we need to make sure that the honest nodes agree on exactly the same value, which is again representative of the honest nodes' inputs. We observe that this problem is related to the Byzantine agreement (BA) [19] and Approximate agreement [17, 25] problems from the literature on distributed systems. While the expected agreement and termination properties are exactly the same as for BA, the validity property coincides with the typical Approximate agreement definition. We call this problem a *distributed oracle agreement* (DORA) problem.

**Definition 1** (Distributed Oracle Agreement— DORA)**.** A distributed oracle agreement (DORA) protocol among $n$ nodes $\{p_1, p_2, \ldots, p_n\}$ with each node having input $v_i$ guarantees the following properties:

**Property 5** (Termination)**.** *All honest nodes eventually decide on some value.*

**Property 6** (Agreement)**.** *The output value $\mathcal{S}$ for all honest nodes is the same.*

**Property 7** ((Min-max) Validity)**.** *The output value is in the convex hull of the honest nodes' inputs. For scalar inputs, this coincides with Property 2: $H_{min} \leq \mathcal{S} \leq H_{max}$*

Similar to the BA problem, it is easy to observe that DORA also requires an honest super majority. It is however interesting to observe that this bound persists even when the oracle nodes have access to the SMR/blockchain service defined in Section 2.3. Unlike a typical broadcast channel, this SMR channel is asynchronous to different receiving nodes. Therefore, it is not possible to differentiate between slow nodes and crashed nodes. The protocol needs to make progress with only $n - f$ inputs, where $f$ out of $n$ nodes are Byzantine. Access to the SMR service is still helpful as it already overcomes the FLP impossibility [22]. We can develop protocols for BA and DORA in a purely asynchronous manner without requiring any distributed randomness (such as common coins) [12].

The requirement of an honest super-majority for DORA can be a scalability issue as the number of oracle nodes increases. For optimistic scenarios, we overcome this issue by making a practical assumption on the input values. In particular, if we assume that inputs from all honest nodes form a coherent cluster within a reasonably small agreement distance $\mathfrak{d}$, then we can solve DORA requiring only an honest majority instead

of an honest super majority. We call this new problem DORA-CC which may offer a slightly weaker validity property stated below:

**Definition 2** (Distributed Oracle Agreement with Coherent Cluster (DORA-CC))**.** A DORA-CC protocol among $n$ nodes $\{p_1, p_2, \ldots, p_n\}$ with each node having input $v_i$ such that these input values form a coherent cluster for a distance $\mathfrak{d}$, guarantees the following property in addition to Termination(Property 5) and Agreement(Property 6) properties.

**Property 8** (Approximate (Min-max) Validity.)**.** *The output $\mathcal{S}$ is within the interval $[\mathcal{H}_{min} - \mathfrak{d}, \mathcal{H}_{max} + \mathfrak{d}]$.*

### 3.3 DORA with Coherent Cluster Protocol

DORA-CC only needs an honest majority, while we expect the tribe to offer honest nodes with a supermajority. In an optimistic scenario, this allows us to only employ a subset of nodes within the tribe. We divide the tribe $N_t$ into multiple mutually exclusive clans of size $|N_c|$, where $N_c$ denotes the set of nodes belonging to a clan $c$. While the number of Byzantine nodes within $N_t$ is restricted to $f_t$, we choose $|N_c|$ such that the number of Byzantine nodes are at most $f_c = \frac{|N_c|-1}{2}$ with a very high probability. Each such clan of size $|N_c|$ can be given the responsibility to emit $\mathcal{S}$-values for different variables. For simplicity, we only focus on a single variable $\tau$ in this paper. The process, however, can be replicated for multiple variables.

There could be times when the inputs from all honest nodes may not form a coherent cluster within the distance $\mathfrak{d}$. For such a scenario, we aim at first identifying this volatility in a distributed fashion and then securely switching to the fallback protocol for the DORA instance over the entire tribe. Now, when we run the DORA instance over the entire tribe, we ensure that the output satisfies Definition 1 and is representative of the existing market conditions.

Additionally, we also uniformly randomly select a *family* $\mathbb{A}$ of nodes from the tribe such that at least one of them is honest with a high probability. These nodes are designated as aggregators as they are supposed to securely collect information from the clan nodes and post it on the SMR. Note that these aggregators are employed solely to reduce the total number of interactions with the blockchain. Since the nodes sign their inputs, there are no safety issues regarding a Byzantine aggregator forging them. If there is only a single aggregator, there is almost $\frac{1}{3}$ probability of it being Byzantine, which may require an aggregator change to ensure the progress of the protocol. Our multi-aggregator model ensures progress without requiring any aggregator changes, thus reducing latency. However, this comes with added communication complexity.

Pseudocode for DORA-CC in the presence of volatile data feeds is given in Algorithm 2. It proceeds as follows:

1. Using Algorithm 1, every node would gather data from $2f_d + 1$ uniformly randomly assigned data sources, compute the median from all the values received and send the median to all the aggregators as a VALUE message. Each node starts a timer with $T_{fallback}$ (Line 6) .

2. An aggregator waits until a coherent cluster of size $f_c + 1$ is formed. Once the cluster is formed, it computes the mean (Line 12). The aggregator then sends the set of VALUE messages that formed a cluster along with the mean as a VPROP message to all the clan nodes (Line 13). This message would convey that the aggregator proposes the $\mu$ to be the $\mathcal{S}_r$.

3. Upon receiving a proposal VPROP with $CC$ and $\mu$ from an aggregator, the node performs a validation. The node would validate that (i) $CC$ contains signed messages from the nodes of the clan, (ii) the values in $CC$ forms a coherent cluster within $\mathfrak{d}$, and (iii) that $\mu$ is indeed the arithmetic mean of the values in $CC$. If the validation is successful, it sends its signed vote VOTEVP$(CC, \mu, r)$ to the aggregator (Lines 15 to 16).

4. In an optimistic scenario, the aggregator would receive $f_c + 1$ votes of the kind VOTEVP$(CC, \mu, r)$ approving the proposal to allow it to form a quorum, prepare quorum certificate $\mathcal{QC}$ and post VPOST$(CC, \mu, r, \mathcal{QC})$ on the SMR (Line 20). The nodes in turn would witness VPOST$(CC, \mu, r, \mathcal{QC})$ on the SMR, agree on $\mu$ as $\mathcal{S}_r$ and conclude the current round (Line 22).

5. In an unusual scenario, a coherent cluster of size $f_c + 1$ can not be formed by any of the aggregators. This could happen either due to extreme volatility during data feed, or due to network asynchrony

**Algorithm 2** *Compute*$\mathcal{S}$ $(p_i, N_c, \mathbb{A}, r)$

**input:** $r$ is the round identifier, $N_c$ is the set if nodes in the clan, $\mathbb{A}$ is the set of aggregators, $p_i$ is the ID of this node
**Upon** init **do**
    $ADS \leftarrow 2f_d + 1$ uniformly randomly assigned data-sources from $DS$
    $O_{p_i} \leftarrow GetDataFeed(ADS)$
    **send** $\mathsf{VALUE}(\mathcal{Q}_{0.5}(O_{p_i}), r)_i$ to all nodes in $\mathbb{A}$
    start timer $T_{fallback}$
    **if** $p_i \in \mathbb{A}$ **then** $\forall_{p_j \in N_c} obs[p_j] \leftarrow \perp$

**Upon** receiving $\mathsf{VALUE}(v, r)_j$ from a node $p_j$ **and** $p_i \in \mathbb{A}$ **do**
    $obs[p_j] \leftarrow v$
    **if** $\exists_{CC \subseteq obs} |CC| \geq f_c + 1$ **then**
        $\mu \leftarrow mean(CC)$
        **send** $\mathsf{VPROP}(CC, \mu, r)_i$ to all nodes in $N_c$

**Upon** receiving $\mathsf{VPROP}(CC, \mu, r)_j$ from $p_j \in \mathbb{A}$ **do**
    **if** $Validate(\mathsf{VPROP}(CC, \mu, r)_j) = true$ **then send** $\mathsf{VOTEVP}(CC, \mu, r)_i$ to $p_j$

**Upon** receiving $\mathsf{VOTEVP}(CC, \mu, r)_j$ from $p_j$ **and** $p_i \in \mathbb{A}$ **do**
    **if** $\mathcal{QC}$ is formed on $\mathsf{VOTEVP}(CC, \mu, r)$ **then** ▷ Quorum with $f_c + 1$ votes
        $\mathsf{post}_{\mathsf{SMR}}(\mathsf{VPOST}(CC, \mu, r, \mathcal{QC}))$

**Upon** witnessing the **first** $\mathsf{VPOST}(CC, \mu, r, \mathcal{QC})$ on $\mathsf{SMR}$ **do**
    $\mathcal{S}_r \leftarrow \mu$
    **return**

**Upon** timeout on $T_{fallback}$ **do**
    **send** $\mathsf{VOTEFT}(fallback, r)_i$ to all nodes in $\mathbb{A}$

**Upon** receiving $\mathsf{VOTEFT}(fallback, r)_j$ from $p_j \in N_c$ **and** $p_i \in \mathbb{A}$ **do**
    **if** $\mathcal{QC}$ is formed on $\mathsf{VOTEFT}(fallback, r)$ **and** $T_{fallback}$ for $p_i$ has already timed out **then**
        $\mathsf{post}_{\mathsf{SMR}}(\mathsf{FTPOST}(fallback, r, \mathcal{QC}))$

**Upon** witnessing $\mathsf{FTPOST}(fallback, r, \mathcal{QC})$ on $\mathsf{SMR}$ **do**
    switch to Fallback protocol

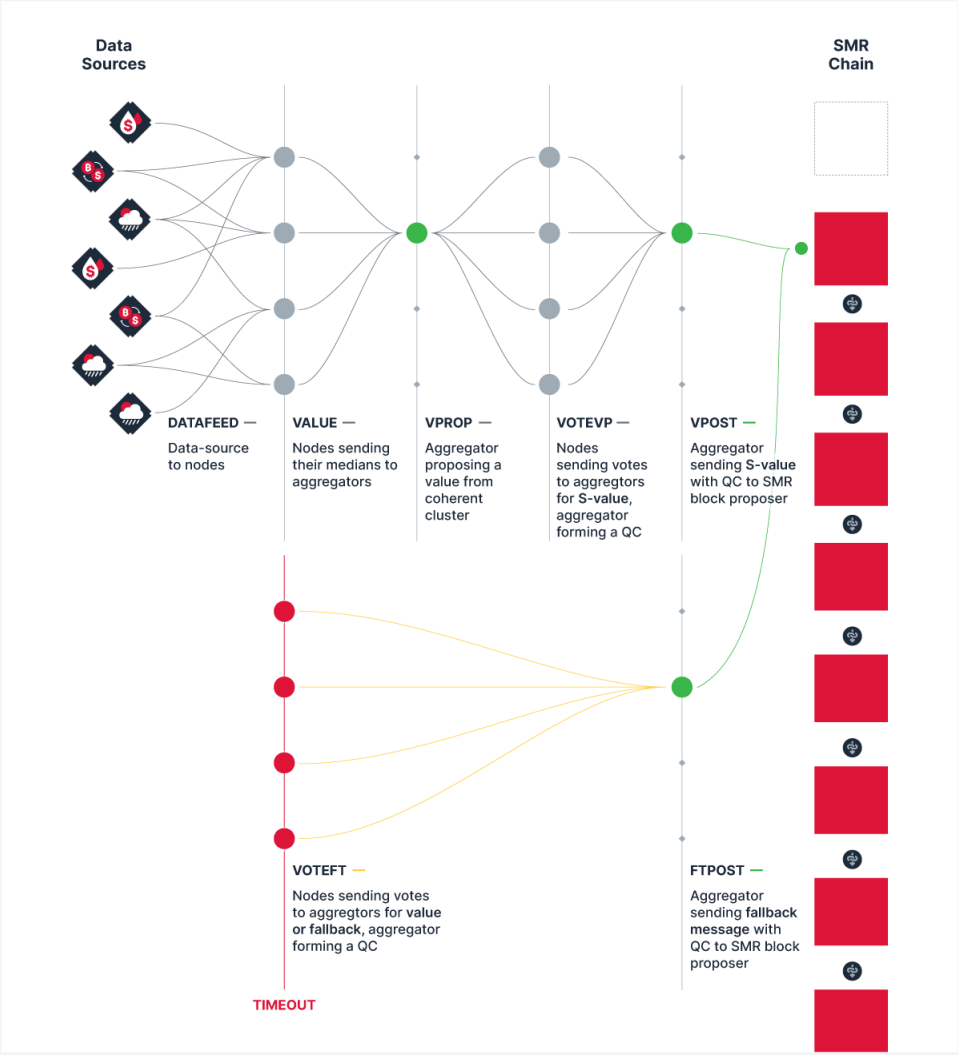



Figure 3: Visualization of DORA protocol

and high network delays. In such a case, the nodes would timeout on $T_{fallback}$ and vote for fallback VOTEFT to all the aggregators (Line 27). This can happen due to high volatility during the data feed window $T_{ds}$ when the honest observations can not form a coherent cluster of size $f_c + 1$. In that case, the aggregators may receive fallback votes VOTEFT from multiple nodes (Line 29). If an aggregator has gathered enough votes for the fallback, it posts the fallback proposal with a quorum certificate on SMR (Line 31).

6. A node witnessing a fallback message FTPOST (due to timeout of $T_{fallback}$) on SMR would switch to the fallback protocol mentioned in Algorithm 3.

**Theorem 2** (Approximate Validity)**.** *The protocol defined in Algorithm 2 if it posts* VPOST$(CC, \mu, r, \mathcal{QC})$ *on* SMR *then* $\mu$ *must be within the interval* $[\mathcal{H}_{min} - \mathfrak{d}, \mathcal{H}_{max} + \mathfrak{d}]$.

*Proof.* There is at least one honest observation within the $f_c + 1$ observations belonging to the cluster $CC$. By definition of a coherent cluster, all other $f_c$ observations, even if reported by Byzantine nodes, have to agree with the honest observation mentioned earlier to form the cluster $CC$. Therefore, no observation can exceed $\mathcal{H}_{max} + \mathfrak{d}$. Similarly, one can argue that no observation can be less than $\mathcal{H}_{min} - \mathfrak{d}$. Therefore, the mean $\mu$ computed by an aggregator must lie within the interval $[\mathcal{H}_{min} - \mathfrak{d}, \mathcal{H}_{max} + \mathfrak{d}]$. □

**Algorithm 3** $FallbackS\ (p_i, r, N_t, \mathbb{A})$

**input:** $p_i$ is the unique id of this node, $r$ is the round identifier, $N_t$ is the set of nodes in the tribe, $\mathbb{A}$ is the set of aggregators
**Upon** init **do**
    $ADS \leftarrow 2f_d + 1$ uniformly randomly assigned data-sources from $DS$
    $O_{p_i} \leftarrow GetDataFeed(ADS)$
    **send** $\mathsf{VALUE}(\mathcal{Q}_{0.5}(O_{p_i}), r)_i$ to all the aggregators (nodes in $\mathbb{A}$)
    **if** $p_i \in \mathbb{A}$ **then**
        $\forall_{p_j \in N_c} obs[p_j] \leftarrow \perp$

**Upon** receiving $\mathsf{VALUE}(v, r)_j$ from $p_j$ **and** $p_i \in \mathbb{A}$ **do**
    $obs[p_j] \leftarrow v$
    $O \leftarrow \{obs[p_j] | p_j \in N_t \wedge obs[p_j] \neq \perp\}$
    **if** $|O| \geq 2f_t + 1$ **then**
        **send** $\mathsf{VPROP}(O, \mathcal{Q}_{0.5}(O), r)_i$ to all nodes in $N_t$

**Upon** receiving $\mathsf{VPROP}(O, v, r)_j$ **and** $p_j \in \mathbb{A}$ **do**
    **if** $Validate(\mathsf{VPROP}(O, v, r)_j) = true$ **then**
        **send** signed vote $\mathsf{VOTEVP}(O, v, r)_i$ to $p_j$

**Upon** receiving $\mathsf{VOTEVP}(O, \mathcal{Q}_{0.5}(O), r)_j$ from $p_j$ in $N_t$ **and** $p_i \in \mathbb{A}$ **do**
    **if** $\mathcal{QC}$ is formed on $\mathsf{VOTEVP}(O, \mathcal{Q}_{0.5}(O), r)$ **then** ▷ Quorum with $2f_t + 1$ votes
        $\mathsf{post}_{\mathsf{SMR}}(\mathsf{VPOST}(O, \mathcal{Q}_{0.5}(O), r, \mathcal{QC}))$

**Upon** witnessing the **first** $\mathsf{VPOST}(O, v, r, \mathcal{QC})$ on SMR **do**
    $\mathcal{S}_r \leftarrow v$
    **return**

The fallback protocol where the entire tribe $N_t$ participates is shown in Algorithm 3. Note that we assume that at most $f_t \leq \frac{|N_t|-1}{3}$ many nodes in the tribe may exhibit Byzantine behaviour.

1. All the nodes gather observations from their assigned data sources and then send corresponding medians to the aggregators (Lines 2 to 7). Note that even the clan nodes that triggered the fallback gather fresh data along with the rest of the tribe.

2. When an aggregator receives a VALUE from a node, it checks whether it has received inputs from $2f_t + 1$ nodes (Line 12). Since at most $f_t$ out of total $|N_t| = 3f_t + 1$ nodes may be Byzantine, by gathering inputs from $2f_t + 1$ votes, the aggregator is assured that within this set of inputs, there is an honest majority. From these inputs, it calculates its median and sends $\mathsf{VPROP}(O, \mathcal{Q}_{0.5}(O), r)$ to all the nodes in the tribe $N_t$ for $\mathcal{Q}_{0.5}(O)$ to be considered as $\mathcal{S}_r$ for the current round $r$ (Line 13).

3. A node receiving a proposal $\mathsf{VPROP}(O, v, r)$ from an aggregator validates that (i) observations in $O$ are signed by nodes in $N_t$, (ii) the value $v$ is indeed the median of $O$, and (iii) the message is indeed from an aggregator in $\mathbb{A}$ (Line 16). It sends back its vote $\mathsf{VOTEVP}(O, \mathcal{Q}_{0.5}(O), r)$ to the same aggregator (Line 17).

4. An aggregator, upon receiving $2f_t+1$ votes, prepares the quorum certificate $\mathcal{QC}$ and posts $\mathsf{VPOST}(O, \mathcal{Q}_{0.5}(O), r, \mathcal{QC})$ on SMR (Line 21).

5. The nodes conclude round $r$ after reaching consensus on $v$ as $\mathcal{S}_r$ when they witness $\mathsf{VPOST}(O, v, r, \mathcal{QC})$ on SMR (Line 24).

It is evident from Theorem 1 that the $\mathcal{S}_r$ reported by the fallback protocol will always be inside the interval $[\mathcal{H}_{min}, \mathcal{H}_{max}]$. Due to multiple aggregators, it is possible that an FTPOST is posted on SMR, but a VPOST by another aggregator emerging from Algorithm 2 is delayed in reaching the SMR. In this case, the nodes would switch to the fallback protocol and then witness the delayed VPOST message. In that case, the $\mathcal{S}$ would be within $[\mathcal{H}_{min} - \mathfrak{d}, \mathcal{H}_{max} + \mathfrak{d}]$ as defined in Property 8. The nodes would only consume the $\mathcal{S}$ from the first VPOST message on SMR for any given round.

**Theorem 3** (Fallback Termination)**.** *Algorithm 3 eventually terminates for all honest nodes if (i) $N_t$ has an honest super majority and (ii) $\mathbb{A}$ has at least one honest aggregator.*

*Proof.* $GetDataFeed()$ terminates within $T_{ds}$ as per the design. All $2f_t + 1$ honest nodes would send their corresponding median values to all the aggregators. Since there is at least one honest aggregator, it would eventually receive values from $2f_t+1$ nodes. It is possible that in the subset $O$ considered by the aggregator, there are some Byzantine observations. However, the aggregator would send $\mathsf{VPROP}(O, \mathcal{Q}_{0.5}(O), r)$ to all the nodes in $N_t$. An honest node can validate the aggregator's proposal, even if its own value is not in $O$. Therefore, an honest aggregator would eventually receive $2f_t+1$ votes on its proposal. This would be posted on SMR by the aggregator, ensuring at least one valid proposal on SMR. Therefore, all the honest nodes would eventually witness some valid $\mathsf{VPOST}(O, v, r, \mathcal{QC})$ on SMR and would reach a consensus on $\mathcal{S}_r$ to be $v$. □

**Theorem 4** (DORA-CC-Fallback Termination)**.** *Algorithm 2 eventually terminates for all honest nodes if (i) $N_c$ has honest majority, (ii) $N_t$ has honest super majority, and (iii) $\mathbb{A}$ has at least one honest aggregator.*

*Proof.* All the honest nodes would finish gathering their data feed within $T_{ds}$. We will prove the termination of round $r$ by all the honest nodes.

An honest node would conclude a round and move to the next round only upon witnessing a $\mathsf{VPOST}(Obs, v, r, \mathcal{QC})$ for some set of observations $Obs$. There are multiple scenarios to be considered here.

1. If at least one of the aggregators is able to form a coherent cluster and post a value via VPOST message on SMR as the first message for round $r$, then all the honest nodes would witness it and conclude round $r$. (Line 20).

2. If the first message for round $r$ that appears on SMR is an FTPOST message (Line 33). In this case, all of the honest nodes would switch to the fallback protocol (Algorithm 3). It is possible that some other aggregator in $\mathbb{A}$ is able to post a value via a VPOST message on SMR after the FTPOST message mentioned earlier. Even in this case, all the honest nodes would witness this VPOST in Algorithm 3 (Line 23) and conclude round $r$. If no VPOST is posted on SMR from Algorithm 2 after an FTPOST, since all the honest nodes would have switched to the fallback protocol, we are guaranteed termination due to Theorem 3.

Therefore, we need to prove that from Algorithm 2 either VPOST or FTPOST will definitely be posted.

1. If an honest aggregator is able to form a coherence cluster, due to an honest majority in the clan, it would be able to get $f_c + 1$ votes approving the cluster and its mean and would be able to send a VPOST to SMR.

2. If no aggregator is able to form a coherent cluster, all the honest nodes would timeout on their $T_{fallback}$ eventually. Therefore, there will be $f_c + 1$ VOTEFT votes will be sent to all the aggregators eventually and some honest aggregator would post FTPOST on SMR.

□

SMR safety (Property 3) ensures that all the nodes eventually observe messages in the exact same total order. The agreement property (Property 6) follows directly from this SMR safety property. Notice that for $n > 2f$ with $(n, n - f)$ threshold signature setup, an aggregator may create multiple signed quorums; however, the SMR safety again helps as the nodes pick the first published quorum.

## 3.4 Mean v/s Median

In Algorithm 2, we reach an agreement within the agreement distance $\mathfrak{d}$ with only $f_c + 1$ nodes. Since it guarantees that there would be at least one honest observation within this set of $f_c + 1$ observations, we decide to use mean to ensure that all the honest observations are definitely included in the final $\mathcal{S}$ being proposed. Note that while using the median of the coherent cluster would still be within the bounds $[\mathcal{H}_{min} - \mathfrak{d}, \mathcal{H}_{max} + \mathfrak{d}]$, a Byzantine majority within the set of $f_c + 1$ observation may move the $\mathcal{S}$ slightly in the favor of the adversary. Using the mean allows honest observations to counterbalance byzantine observations, thereby mitigating a malicious movement of $\mathcal{S}$.

On the other hand, Algorithm 3 uses the median as an aggregation since it offers maximum the robustness and resilience in the presence of corrupt data due to its breakdown point being 0.5. Since there is no notion of the observations forming a coherent cluster in Algorithm 3, using the mean would allow an adversary to move $\mathcal{S}$ in its favour arbitrarily and in an unbounded fashion.

### 3.4.1 Analysis of Communication Complexity

Let us analyze how many messages and bits need to be transmitted by Algorithm 2 and Algorithm 3. We shall use $n_c = |N_c|$ to denote the size of the clan, $n_a = |\mathbb{A}|$ to denote the size of the family of aggregators, and $n_t = |N_t|$ to denote the size of the tribe.

To obtain data from the data sources, we need $(2f_d + 1)n_c$ messages. Nodes would then send $n_a n_c$ VALUE messages to aggregators. After aggregation, the aggregators would send $n_c n_a$ VPROP messages. The nodes would generate $n_c n_a$ VOTEVP messages to be sent to aggregators which would subsequently send at most $n_a$ messages to the SMR. This would result in the total number of messages being transmitted by Algorithm 2 to $(2f_d + 1)n_c + 3n_c n_a + n_a$ when a coherent cluster is formed. Considering $n_c n_a$ to be the dominating term, the message complexity would be $O(n_c n_a)$ in the optimistic scenario. Let the total number of variables for which we need to run DORA be denoted as $n_\tau$. If we run DORA for different $\tau$ independently, then the message complexity would increase to $O(n_c n_a n_\tau)$. However, one can batch DORA messages for different $\tau$ together in which case the message complexity would again reduce to $O(n_c n_a)$.

In the case when $T_{fallback}$ times out, the nodes directly send votes to initiate the fallback. This would require $O(n_c n_a)$ messages at the most. Though each aggregator may try to post FTPOST to SMR, only one can succeed and the other will be discarded as a duplicate by SMR. Since each aggregator may transmit

this message, it should be counted, thus, requiring a total of $O(n_a)$ messages. When the nodes switch to the fallback protocol, the data feed now requires $O((2f_d+1)n_t)$ messages. There would be $O(n_t n_a)$ VALUE messages by the nodes, $O(n_t n_a)$ VPROP messages by the aggregators and $O(n_t n_a)$ VOTEVP messages by the nodes. The aggregators would at most transmit $O(n_a)$ messages to the SMR. Therefore, the total number of messages in a given round is $O(n_t n_a)$, since $n_t > n_c$.

For analyzing complexity in terms of the number of bits transmitted, we need to take into account the size of each message. Let the length of the data-feed message and VALUE message and hash of any messages be upper-bounded by $k$. We assume that within this length we can store all the information needed for the protocol such as a round identifier, a node identifier, a data-source identifier, a $\tau$ value etc. The data feed stage would require $O(k(2f_d+1)n_c)$ bits. Let us assume $\lambda$ to be the length of a signature. There would be $O((k+\lambda)n_c n_a)$ bits required for the nodes to send VALUE messages to aggregators. Since the aggregator sends back not only the $\mu$ but also the set of original VALUE messages forming a coherent cluster, the number of bits required for VPROP messages would be $O(((k+\lambda)n_c + k + \lambda)n_c n_a)$. The nodes send their votes on a value proposal, thus requiring $O((k+\lambda)n_c n_a)$ bits for VOTEVP messages. We assume that the nodes would sign the hash of the VPROP message received earlier and return it as an approval. The aggregators then form a $\mathcal{QC}$ and send it to the SMR, which would require $O((k+\lambda n_c)n_a)$ bits of transmission since each $\mathcal{QC}$ would have its size in proportion to $n_c$. Therefore, the communication complexity in bits would be $O((k+\lambda)n_c^2 n_a)$ for DORA-CC. For $n_\tau$ many variables it would be $O((k+\lambda)n_c^2 n_a n_\tau)$.

In case the fallback happens, the communication complexity in terms of bits would be similar to the one described above, but now the messages would be transmitted at a tribe level. Thus the number of bits required for a given round would be $O((k+\lambda)n_t^2 n_a)$ for a single $\tau$ and $O((k+\lambda)n_t^2 n_a n_\tau)$ for $n_\tau$ variables.

### 3.4.2 Error analysis

When the number of Byzantine observations is $f$ and we consider $2f+1$ observations, then we know that as per Theorem 1 the median value will fall within the bounds defined by $\mathcal{H}_{min}$ and $\mathcal{H}_{max}$. However, this is not the case when we only consider $f+1$ observations. A Byzantine aggregator could find $\mathcal{H}_{max}$ and insert $f_c$ Byzantine observations with value $\mathcal{H}_{max}+\mathfrak{d}$ to form a cluster. It is possible that all the other $f_c$ honest observations have the value $\mathcal{H}_{min}$. Had only the honest observations been considered to form a cluster, the value of $\mu$ would have been $\frac{f_c\mathcal{H}_{min}+\mathcal{H}_{max}}{f_c+1}$ but instead, we would end up with $\mu = \frac{\mathcal{H}_{max}+f_c(\mathcal{H}_{max}+\mathfrak{d})}{f_c+1}$. Therefore, we would have an error upper bound of $\|\mathcal{H}_{min}-\mathcal{H}_{max}\|_1+\mathfrak{d}$ when Algorithm 2 emits $\mathcal{S}_r$ via the DORA-CC protocol.

In the case of Algorithm 2, it may be possible that $\|\mathcal{H}_{min}-\mathcal{H}_{max}\|_1 \leq \mathfrak{d}$ does not hold and an honest aggregator would have proposed a fallback. However, a Byzantine aggregator may be successful in preventing the fallback by forming a cluster of one honest observation $\mathcal{H}_{max}$ with $f_c$ Byzantine observations with value $\mathcal{H}_{max}+\mathfrak{d}$. Had the protocol switched to the fallback protocol, the smallest value of the median produced by Algorithm 3 would have been $\mathcal{H}_{min}$. Therefore, even in this case, the upper bound for the error would be $\|\mathcal{H}_{min}-\mathcal{H}_{max}\|_1+\mathfrak{d}$. The arguments with respect to Byzantine observations forming a cluster with $\mathcal{H}_{min}$ would be symmetric and do not result in any change in the error upper bound.

**Theorem 5** (Lower bound on error upper bound)**.** *A protocol for agreeing on a $\mathcal{S}$, where $3f_t+1$ nodes participate out of which $f_t$ of these nodes could be Byzantine, and a median of values from non-deterministically chosen[3] $2f_t+1$ nodes is proposed as the $\mathcal{S}$, would have an error upper bound of at least $\|\mathcal{H}_{min}-\mathcal{H}_{max}\|$.*

*Proof.* Out of the total $3f_t+1$ nodes, $f_t$ could be Byzantine. Out of $2f_t+1$ honest observations, let $2f_t$ of them have the value $\mathcal{H}_{min}$ with one honest observation having the value $\mathcal{H}_{max}$. Let all the Byzantine observations have the value $\mathcal{H}_{max}+c$, where $c>0$. Out of these $3f_t+1$ observations, if the median is calculated from only $2f_t+1$ honest observations, the median would be $\mathcal{H}_{min}$. Instead, if the median is calculated from $2f_t+1$ values where $f_t$ honest values are $\mathcal{H}_{min}$, one honest value is $\mathcal{H}_{max}$ and all the Byzantine observations have the value $\mathcal{H}_{max}+c$ then the median would be $\mathcal{H}_{max}$. Therefore, for any such protocol, the largest possible error can not be less than $\|\mathcal{H}_{min}-\mathcal{H}_{max}\|_1$. □

Note that Theorem 5 would hold for any protocol, that computes a median of only $2f_t+1$ values out of the total $3f_t+1$ values that may be available.

[3]The non-determinism choice is introduced due to the non-determinism in network delay

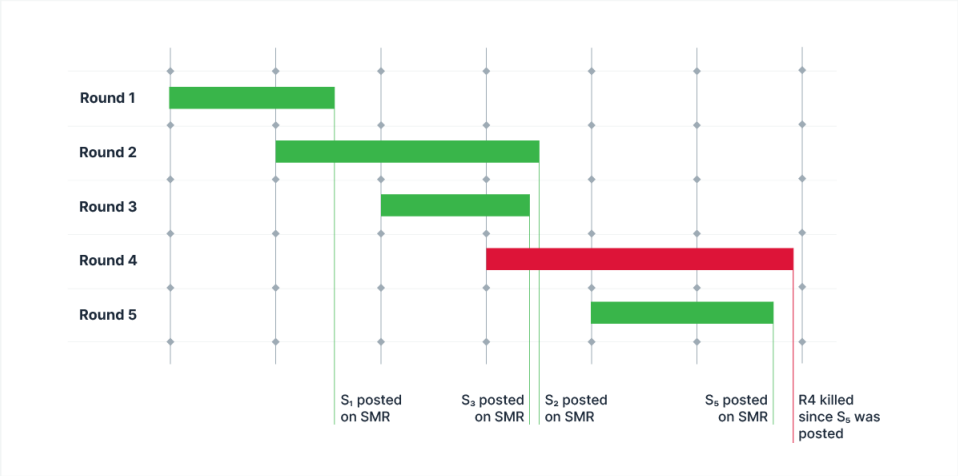



Figure 4: Tick-start DORA timeline

It is evident from Theorem 5 that Algorithm 2 may increase the error upper bound by only $\mathfrak{d}$.

### 3.4.3 Error compounding

Since it is possible that Byzantine actors may introduce an error of $\|\mathcal{H}_{min} - \mathcal{H}_{max}\|_1 + \mathfrak{d}$, one can wonder if a Byzantine aggregator across several successive rounds can compound the error. This is not possible because an $\mathcal{S}_r$ is bounded by $\mathcal{H}_{min}$ and $\mathcal{H}_{max}$ of honest observations from the round $r$ only. Therefore, the protocols in Algorithm 2 or Algorithm 3 do not allow the error to be compounded in successive rounds.

## 3.5 Tick-start DORA

Readers should note that Algorithm 2 and Algorithm 3 do not have any dependence on earlier rounds. Therefore, the protocol for every round of agreement can be initiated and executed completely independently.

One can therefore envisage a version of DORA protocol, which initiates a round at a regular tick, with the interval between two consecutive ticks being pre-determined (e.g., 30 seconds or 30 seconds). The underlying communication is partial-synchronous, so it provides no guarantee that a round $r$ would complete before round $r+1$ starts. The advantage of rounds being independent of one another is that one round does not have to wait for earlier rounds to finish.

In normal operating conditions, this may result in values of every round being published sequentially and at somewhat regular intervals. However, due to the vagaries of network delays and abnormal situations, it is possible that $S_r$ for round $r$ gets published on SMR before the value $S_{r-x}$ for round $r-x$, for any $x > 0$. Thus, the values on SMR may appear out of order. For any two rounds $r_1 < r_2$, $S_{r_2}$ is likely to be a fresher value as compared to $S_{r_1}$ since the data gathering for $r_1$ most likely would have started before the data gathering for $r_2$. In such cases, consumers of these values may choose to only consume values from a monotonically increasing subsequence of rounds. For example, as shown in Figure 4 if the sequence of $\mathcal{S}$ being posted on SMR is $\mathcal{S}_1, \mathcal{S}_3, \mathcal{S}_2, \mathcal{S}_5$ then a consumer may only consume $\mathcal{S}_1, \mathcal{S}_3, \mathcal{S}_5$ since using $\mathcal{S}_2$ may serve no practical purpose after having witnessed $\mathcal{S}_3$. Note that the values can appear out of order on SMR, since even after $\mathcal{QC}$ is formed on values by the aggregators, one value may be posted faster than the other on the SMR due to unpredictable delays. If, however, a value from a later round (round 5 as shown in Figure 4) appears on SMR before $\mathcal{QC}$ is formed on a value for a given round (round 4 as shown in Figure 4), then the ongoing round can be aborted or killed.

Since consuming values from older rounds may not serve any practical purpose, there is merit in terminating an older round if an $\mathcal{S}$-value from a later round is seen on the SMR. For example, for round $r$ if the coherent cluster can not be formed resulting in a fallback, completion of the round $r$ may take longer since now the entire tribe has to be engaged to reach an agreement on the value of $\mathcal{S}_r$. Meanwhile, if a coherent cluster is formed in round $r+1$ and $S_{r+1}$ gets posted on the SMR, there is no advantage for nodes

to continue with round $r$ since a *fresh* value is already available on the SMR. Therefore in this case, we change the protocol such that upon witnessing a value $S_r$ from round $r$ on the SMR, all the honest nodes agree to terminate all the rounds $r' <= r$ irrespective of whether a quorum certificate for a value for round $r'$ has been formed or not.

# 4 Analysis

## 4.1 Empirical Analysis of Agreement Distance

We would describe some empirical analysis and simulations of our protocol done on real-world data.

We obtained data for BTC prices in USD from 7 different exchanges, namely, Binance, Coinbase, crypto.com, FTX, Huobi, OKCoin, and OKEx from 1-Oct-2022 to 10-Nov-2022. We divided this data into two parts. The first part consists of the data from 1-Oct-2022 to 10-Oct-2022 which was used in determining various values for $\eth$. The second part consists of the data from 11-Oct-2022 to 10-Nov-2022 including the turbulent FTX collapse period, which was used to simulate the protocol with various values of $\eth$.

The data from 1-Oct-2022 to 10-Oct-2022 was divided into (i) 30 second windows, and (ii) 60 second windows. For each window, the median and the mean were calculated from all the values/observations available within that window. We observed that the mean of these means and the mean of the corresponding medians were less than \$0.02 away from one another. For 30 second and 60 second windows, the mean of means was around \$19605 and \$19606 respectively. Therefore, we used \$19605.5 as the representative price of BTC for the duration of 1-Oct-2022 and 10-Oct-2022.

The simulations were done with $\eth$ set to various values from 0.02% to 0.55% of \$19605.5, the representative price of BTC calculated as described above.

For simulation, we used 30 and 60 seconds as two different values for $T_{ds}$. The simulation data from 11-Oct-2022 and 10-Nov-2022 was divided into (i) 30 second windows, and (ii) 60 second windows. Therefore, every node obtained its data from data sources within the same window for a given round of oracle agreement. We assumed that the nodes of the oracle network have clock-drift within a few hundred milliseconds. [20] We simulated the behavior of 7 nodes, with each node randomly assigned 5 out of the 7 exchanges. Data from each window was used to simulate one round of agreement.

The availability of data from various exchanges is shown in Table 1. Note that for 4 out of 7 exchanges, the data availability is below 50% for the simulation duration of 11-Oct-2022 to 10-Nov-2022. This low availability could perhaps be due to (i) inherent low data availability from exchanges during this period, or (ii) due to gaps in the data gathering/archival processes on our end, or (iii) due to a combination of both. In any case, this allows us to simulate our protocol with low data availability. Our design choice of using multiple data sources per node is justified to improve the reliability of our protocol even when the data availability is low. Notice that for OKCoin we had a value at almost every minute, therefore, while for 30 second windows, it had values for almost every alternate window, whereas for 60 second windows its data availability was very high.

Table 1: Percentage of times we did not have any value from various exchanges for a time window.

| Exchange | Null value percentage for 30 sec windows | Null value percentage for 60 sec windows |
|---|---|---|
| Binance | 51.65 | 52.66 |
| crypto.com | 78.88 | 75.71 |
| Coinbase | 97.02 | 96.31 |
| FTX | 73.87 | 74.12 |
| Huobi | 44.45 | 45.51 |
| OKCoin | 46.66 | 0.5 |
| OKEx | 26.84 | 28.33 |

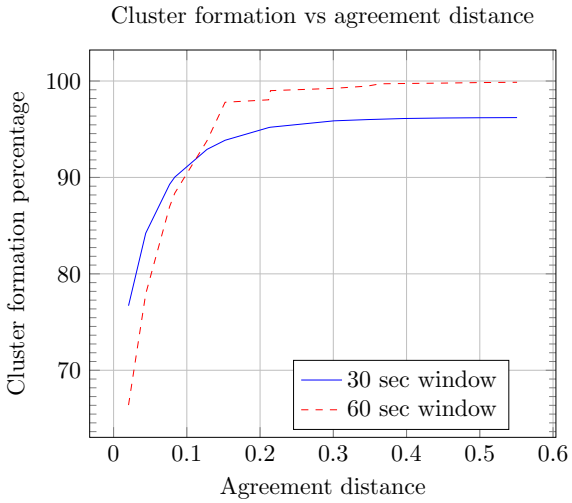



Figure 5: Percentage of times a coherent cluster was formed as agreement distance increased. Agreement distance is displayed as a percentage of 19605, the average BTC price.

For every round, every node would compute the median from the values it obtained from its 5 assigned exchanges. If an exchange had multiple values within the given window, only the latest would be considered by the node. Once every node had computed its median, these medians would be sorted to see if any 4 of the 7 nodes form a coherent cluster for a given value of $\mathfrak{d}$. Figure 5 shows the results from our simulation. As the size of the observation window increases, one would naturally expect to see more deviation in values among values of various exchanges. Therefore, one would expect that for a given $\mathfrak{d}$, the nodes would be able to form coherent clusters more often for 30 second windows as compared to 60 second windows. With $\mathfrak{d}$ of \$16, the cluster formation was achieved 90% and 88% for 30 and 60 seconds windows respectively. However, note the drastic increase in data availability from OKCoin from the 30 to the 60 seconds observation periods. This has contributed to the increased percentage of coherent cluster formation for a 60 second window for larger $\mathfrak{d}$. For example with $\mathfrak{d}$ being \$53, cluster formation is achieved 99% of the time for 60 seconds windows.

Note that the choice of $\mathfrak{d}$ has a bearing on both the safety and the performance of the protocol. Smaller values of $\mathfrak{d}$ would result in the protocol having to fall back more often, whereas higher values of $\mathfrak{d}$ potentially allow higher deviation of $\mathcal{S}$ from the ideal representative value of the mean of honest values.

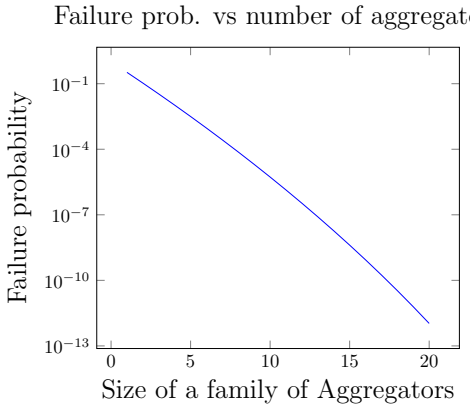
If the value of a $\tau$ increases or decreases significantly, for safety and performance reasons, $\mathfrak{d}$ should be adjusted up and downwards accordingly.

## 4.2 Theoretical Analysis of Probabilistic Safety

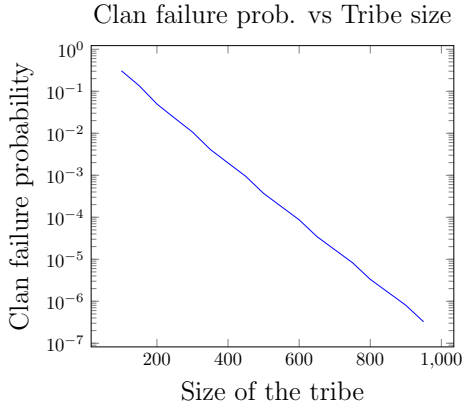
We mention in Section 3.3 that we employ multiple aggregators. Figure 6a shows a logarithmic plot of how the probability of having an entire family consisting of Byzantine aggregators reduces as we increase the size of the family of aggregators. Since $\frac{1}{3}$ of the nodes in the tribe could be Byzantine, it is evident that as we increase the size of the family of aggregators, the probability of not having a single honest aggregator drops exponentially fast.

Figure 6b shows how the probability of having at least one clan with a Byzantine majority changes as we increase the size of the tribe. In the logarithmic plot, we can observe that increasing the size of the tribe to a few hundred nodes would ensure that, with a very high probability, none of the 5 clans randomly drawn from the tribe would have a Byzantine majority. To fully exploit the ability of DORA to tolerate up to 49% Byzantine nodes, we propose to employ probabilistic safety guarantees. Figure 6b provides guidelines on how many total nodes would be required to achieve the safety property with a very high probability.

Note that, if the adversary is fully adaptive and can compromise nodes after families and clans are formed, the above probabilistic safety analysis is not applicable. We expect clans and families to be updated regularly in practice.




(a) Probability of having no honest aggregators in the family of aggregators as the size of the family increases. Here the total nodes are 100 and Byzantine nodes are 33.




(b) Probability of having at least one clan with a Byzantine majority as the size of the tribe increases. Here the number of clans are 5 and Byzantine nodes are 33%.

Figure 6: Probabilistic Safety Analysis

# 5 Extensions

## 5.1 Assigning weights to the data sources

In Algorithm 1, we mandate the nodes listen to $2f_d + 1$ data sources, since $f_d$ of these data sources can turn Byzantine. Each node computes the median of all the observations it makes from these $2f_d + 1$ data sources. This algorithm, however, treats every data source equally. Different data sources, however, may have different reliability guarantees, trading volumes, security guarantees and best practices, etc. It makes sense that when we look at these aspects, different data sources should be treated differently. For example, a data source that provides better reliability guarantees and follows the best security practices should be given more weight when compared to some other data sources which may be inferior in these parameters.

To accommodate the differing traits of different data sources, we can define a weight function $w : DS \to \mathbb{N}^+$, which assigns different positive integer weights to different data sources. The weight can be determined by considering parameters like historical reliability, volume/scale, and security practices followed by the data source etc., and a suitable value can be assigned through data governance. It is important to note that while we assign weight to the data sources, a data source crash or malfunction would affect any function that takes its inputs in proportion to its weight. Therefore, we now must discuss the total weight of the data sources that may turn Byzantine. In this modified setting, we can think of $f_d$ as the sum of the weights of the data sources that can turn Byzantine. To accommodate this change, Algorithm 1 can be modified where every node listens to data sources such that their total weight is $2f_d + 1$. In this case, whenever a value $v$ is received from a data-source $ds$, we would store as many as $w(ds)$ copies in the multi-set $obs$ (Line 7). The size of $Obs$ would still be between $f_d + 1$ and $2f_d + 1$, the lower bound on the number of observations from honest data sources would still be $f_d + 1$ and the number of observations from Byzantine data sources would be upper-bounded by $f_d$. Theorem 1 would still hold in this case. The rest of the protocol for DORA would remain the same.

## 5.2 Cross-correlation across $\tau$

In this paper, we have provided a protocol for solving the DORA problem for a given variable $\tau$. In practice, there are several different variables for which we want to solve the DORA problem for providing oracle services. Therefore, we can leverage relations that exist amongst different variables. For example, let $\tau_1$,$\tau_2$ and $\tau_3$ denote values of `BTC/USD`, `ETH/USD`, and `BTC/ETH`, respectively. We know that under normal circumstances we have the relation $\tau_1 = \tau_2\tau_3$ among these variables. Ideally, we have that $\tau_1 - \tau_2\tau_3 = 0$. In practice, however, we would hope that $\|\tau_1 - \tau_2\tau_3\|_1 \leq \mathfrak{D}_r$, where $\mathfrak{D}_r$ is the *correlation distance* denoting the reasonably small deviation one can expect under normal circumstances.

We can utilize the relations amongst variables to detect anomalous behaviors in the variables. For example, we know that whenever we detect $\|\tau_1 - \tau_2\tau_3\|_1 > \mathfrak{D}_r$, we know that one or more variables are definitely exhibiting anomalous behaviors. For example, let us assume that we have data sources providing data about multiple $\tau$. Let for $ds_i$, we have that $\tau_1 - \tau_2\tau_3 = c_i$. If we have that distances between $c_1$ and other $c_i$ are very large, while $c_i, i \neq 1$ are very close to each other, it would indicate that $ds_1$ is very likely behaving in an anomalous fashion. When we use nodes to perform DORA for values of multiple variables $\tau$, we can use similar multi-variate analyses to detect if one or more nodes are behaving anomalously.

In addition, this multi-variate setting can give us yet another way to solve the DORA problem. In this paper, we have talked about an observation as a scalar value. Instead, one can envisage an observation of a node to be a vector. In this case, we would redefine the agreement of two observations $o_1$ and $o_2$ as their $L_2$-norm distance to be within $\mathfrak{d}$, i.e., $\|o_1 - o_2\|_2 \leq \mathfrak{d}$. This gives rise to a multi-dimensional version of the DORA problem.

## 5.3 Indicators for volatility

High volatility in the sequence of emitted $\mathcal{S}$ could indicate issues either within the oracle network, the data sources, or some other issues (i.e., economic issues if the variable represents a token, stock, or commodity price). There are multiple ways to detect high volatility in the price of a commodity.

One of the popular predicates that is used looks like the following: $\frac{|S_r - S_{r-1}|}{S_{r-1}} > cbthr$, here, $cbthr$ is

a *circuit breaker threshold*. Essentially, whenever a new value deviates from an old value by more than a certain percentage, a circuit breaker is applied which can be used to flag an anomaly or freeze trades.

Another predicate that could be used would be to check if $S_r$ falls within the bounds $S_{r-1} \pm c_h \sigma_h$, where, $c_h$ is a constant parameter and $\sigma_h$ is the standard deviation of the last $W_h$ values. Here, $W_h$ is the size of the *history window* and would be a parameter. Compared to the circuit breaker function mentioned earlier, this function looks at a larger history and tries to determine if the new value deviates from the old value *by more than usual*. The interval defined in this fashion widens and narrows dynamically depending upon the historical deviations observed. This way, if large deviations are sustained, the predicate adapts it to classify it as the norm rather than an anomaly. We call this predicate a *history consistency check*.

As a value addition to the consumer of $\mathcal{S}$, it is very easy to flag a value of $\mathcal{S}_r$ if it fails the history consistency check. In a more general setting, one can think about a series of constant parameters $0 < c_1 < c_2$ such that if the $\mathcal{S}$ falls within $S_{r-1} \pm c_{1_h} \sigma_h$, it is flagged as Green. If not, but if it falls within $S_{r-1} \pm c_{2_h} \sigma_h$, it may be flagged as Yellow. If it falls outside of $S_{r-1} \pm c_{2_h} \sigma_h$, it may be flagged as Red. In general, one can define a volatility index based on some monotonically increasing function of $\sigma_h$ and $c_h$.

It may be left to the consumer of this information to take such flags or index into account based on the use case and the application. For example, one may not wish to carry out or trigger certain financial transactions during the times of high volatility. It is possible to let the oracle protocol simply produce a sequence of values, and any such flagging, filtering, or index decoration may be accomplished as post-processing by a smart contract.

# 6 Discussion

## 6.1 Size of the oracle network

It is crucial to understand that the size of the oracle network has a huge bearing on both the safety and the liveness of DORA. Two different oracle networks $A$ and $B$, both operating with $3f + 1$ nodes, where $f$ of them could be Byzantine, may differ greatly in offering safety and liveness. For example, let us assume that $A$ is running with 10 nodes and $B$ is running the same protocol with 1000 nodes.

- With fewer nodes, the likelihood of nodes spread across different geographies, different data centers, different countries, and different ISPs (*Internet Service Providers*) is far lower when compared to a network operating with more nodes, when the nodes are independent entities. Internet traffic being blocked by a country or an ISP, a crash or a compromise of a data center are events that have the potential to cause a large fraction of nodes being disconnected from the oracle network. In such extreme events, the likelihood that $A$ would have more than $f$ nodes getting disconnected is much higher as compared to network $B$.

- Let us assume that it takes $\$1M$ to *bribe* a node operator to behave in a malicious fashion. Even if we consider a linear model for the amount of bribe required to make a node malicious, it requires $\$4M$ to compromise the safety of network $A$ whereas one would have to gather $\$334M$ to compromise the safety of network $B$. In a proof-of-stake framework, participant nodes are required to stake some amount to be able to participate. In such a framework, let there be a reward for a whistle-blower which can prove that $f_c + 1$ nodes deviated from the protocol which resulted in an *incorrect* value $S_r$ being emitted. Let such a reward be equivalent to the stake of all $f_c + 1$ nodes. In such a case, the amount of the bribe required increases in a quadratic fashion since each node has to be bribed more than the stake of 51% of the nodes. Due to this quadratic increase in the amount of bribe, the adversary needs a much greater amount to compromise network $B$ in comparison to network $A$.

- Let us assume that an adversary is using malware in order to execute its malicious plan of compromising oracle safety. Irrespective of whether the rate of spread of malware is linear or exponential, it requires much more time to affect 34% of the nodes of network $B$ as compared to network $A$.

In general, for almost any attack vector, it would require significantly more resources (i.e., time, money etc.) to compromise a larger network as compared to a smaller network.

## 6.2 Data source failure

There are some other oracle models which are opaque about how the data is obtained by the node from a data source. While the probability that the data source itself is malicious may be very low in practice, a transient glitch in the data source is a real possibility that one must be able to accommodate while designing an oracle with strong safety guarantees. There are plenty of documented glitches and bugs [6, 34] which have resulted in stock exchanges coming to a grinding halt, or worse, emitting a price of a stock that is far from its actual value.

Imagine a model where one node is receiving its data from one data source. Even though an honest node is relaying the information that it received from the data source as-is to the oracle network, a glitch in the data source can make an honest node behave in a Byzantine fashion. Any oracle design that does not take into account the possibility of a Byzantine data source can not guarantee safety or liveness even in a $3f + 1$ setting, where $f$ nodes of the oracle can turn Byzantine. Imagine a network with $f$ Byzantine nodes and one Byzantine data source feeding to an honest node. This essentially breaches the limit of $f$ nodes behaving in a Byzantine fashion. The bounds of $[\mathcal{H}_{min}, \mathcal{H}_{max}]$ are tight and a single additional Byzantine observation can potentially introduce an unbounded error in the reported value. Our design for DORA ensures that every node is listening to a sufficient number of data sources such that Byzantine behaviour of up to $f_d$ data sources can be tolerated.

When an oracle design does not mandate which data source a node should receive its data from, it is likely that many nodes would prefer receiving their data from a data source that is fast, considered relatively more reliable, or cheaper to subscribe to, etc. This can skew the distribution of nodes listening to data sources in favour of a few data sources. If one node is listening to only one data source and there is a data source that is feeding data to more than $f$ nodes, this data source becomes a single point of failure for the entire oracle network. Therefore, it is of the utmost importance that every node not only listens to multiple data sources but the mapping of nodes to data sources is fairly uniform to provide higher resilience to the potential failure of a small set of data sources. Due to this consideration, in our oracle design we mandate not only the minimum number of data sources that a node must receive its data from, but also a uniform random assignment of data sources generated via a VRF (*Verifiable Random Function*) that a node must listen to. Such a design results in one node listening to multiple data sources and also, every data source feeding to multiple nodes. Such uniform random assignment practically eliminates the possibility of Byzantine data sources making an honest node behave in a Byzantine fashion.

## 6.3 Role of randomization

When mapping data source to nodes, as well as mapping from nodes to $\tau$, remains static for longer or potentially indefinite period of time, it allows nodes to collude and launch a successful attack that may violate the safety guarantees of the oracle network.

In the oracle design we propose, we create randomly drawn mutually exclusive clans $N_c$ from the tribe $N_t$. If our oracle network has a responsibility to produce a representative value for a set $V$ consisting of multiple distinct variables $\tau$, we divide $V$ into $\lfloor \frac{|N_t|}{|N_c|} \rfloor$ parts and randomly assign each part to a clan, where each clan is responsible for all the variables $\tau$ belonging to its assigned partition. Mapping from the clan to partition is randomly permuted after $T_{rotate}$, which we shall term it as a *rotation*. Additionally, after $T_{shfl}$, complete reorganization and shuffling of nodes of $N_t$ into various $N_c$ happens, which shall be termed as a *shuffle*.

An additional possibility is to keep the size of the tribe fixed at $|N_t|$. Therefore, after every $T_{churn}$, a small fraction of nodes are churned out from the tribe and new nodes wanting to join the tribe can replace them. A node that is churned out has to mandatorily execute a VDF (*Verifiable Delay Function*) before it can be considered to rejoin the tribe.

Let us assume that Byzantine nodes within a particular clan $N_c$ are colluding. Let us assume that they want to manipulate the value of a specific variable $\tau$, say `BTC/USD`. Note that the assignment of a variable $\tau$ to a clan is not under the control of the adversary and such an assignment is done through a VRF. Therefore, any such manipulation that the adversary wishes to cause is limited to the time duration $T_{rotate}$ after which they will be given responsibility of a different variable. Note that, even within a clan, a Byzantine node can introduce an additional deviation of at most $\mathfrak{d}$. In practice, $\mathfrak{d}$ should be kept smaller than the average

deviation in the value of $\tau$ while making a trade. Additionally, the shuffling of nodes after $T_{shfl}$ ensures that any such clan-level collusion is broken down.

Let us now consider a case in which all the Byzantine nodes of the entire tribe $N_t$ are colluding. Please note that as described in Section 6.1, the resources required by an adversary to compromise a sufficient fraction of a network increases as the size of the network increases. Even with global collusion, the adversary can not manipulate a $\tau$ unless the distribution of the nodes of $N_t$ amongst clans are such that some clan has Byzantine majority. Churning out of randomly selected nodes also acts as a mitigating factor in breaking up any node collusion.

Moreover, we can leverage the idea of cross-correlations that exist amongst various $\tau$ (Section 5.2). We can ensure that when an invariant involving multiple variables, say $\tau_1 - \tau_2\tau_3 = 0$, must hold then $\tau_1$, $\tau_2$, and $\tau_3$ are assigned to different clans. This makes it almost impractical for any adversary to remain undetected in a forensic analysis of data. In the future, we would explore the idea of the SMR doing an additional check that the values being posted by an aggregator for a variable keeps the invariant involving that variable within a tolerable range. This is one potential idea where cross-correlation can act as a preventive measure as opposed to an investigative measure.

# 7 Conclusion

We present a novel distributed oracle agreement protocol that allows the oracle network to function with only $2f + 1$ nodes, when the prices of a commodity are not fluctuating wildly, by leveraging SMR as an ordering primitive and updating the notion of *agreement* amongst nodes.

We have shown that data sources do pose a data availability risk and therefore it is wise to mandate nodes to gather data from multiple data sources. We have also shown the trade-off that the agreement distance parameter offers in terms of safety and performance.

We show that it is possible to build safe and efficient oracle networks with high probabilistic safety guarantees by making appropriate choices on the size of the network and the size of the family of aggregators.

# References


[1] `https://www.investopedia.com/what-went-wrong-with-ftx-6828447`.

[2] Provable blockchain oracle. `https://provable.xyz`, 2022.

[3] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *ACM PODC*, page 337–346, 2019.

[4] Hamda Al-Breiki, Muhammad Habib Ur Rehman, Khaled Salah, and Davor Svetinovic. Trustworthy blockchain oracles: Review, comparison, and open research challenges. *IEEE Access*, 8:85675–85685, 2020.

[5] Renas Bacho and Julian Loss. On the adaptive security of the threshold BLS signature scheme. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACMCCS*, pages 193–207, 2022.

[6] Dr. Johannes Bohnet. `https://www.seerene.com/news-research/software-errors-destroy-millions-of-dollars-every-second-0`, 2015. [Online: accessed on 21-Sep-2022].

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, volume 2248, pages 514–532, 2001.

[8] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[9] Lorenz Breidenbach, Christian Cachin, Alex Coventry, Ari Juels, and Andrew Miller. Chainlink off-chain reporting protocol. Technical report, Chainlink Labs, 2021.

[10] Business Insider. Coinmarketcap glitch lists bitcoin at $799 billion. markets.businessinsider.com link, 2022.

[11] Christian Cachin, Daniel Collins, Tyler Crain, and Vincent Gramoli. Anonymity preserving byzantine vector consensus. In *ESORICS*, page 133–152, 2020.

[12] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptol.*, 18(3):219–246, 2005.

[13] Srdjan Capkun, Ercan Ozturk, Gene Tsudik, and Karl Wüst. Rosen: Robust and selective non-repudiation (for tls). In *Cloud Computing Security Workshop (CCSW)*, page 97–109, 2021.

[14] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *ACM CCS*, pages 719–728, 2017.

[15] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. On the (limited) power of non-equivocation. In Darek Kowalski and Alessandro Panconesi, editors, *PODC*, pages 301–308, 2012.

[16] Coin Telegraph. Defi protocols declare losses as attackers exploit luna price feed discrepancy. cointelegraph.com/news Link, 2022.

[17] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, may 1986.

[18] Assia Doudou and André Schiper. Muteness detectors for consensus with byzantine processes. In *PODC*, page 315, 1998.

[19] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, apr 1988.

[20] Engineering at Meta. Building a more accurate time service at facebook scale. `https://engineering.fb.com/2020/03/18/production-engineering/ntp-service/`, 2020.

[21] Shayan Eskandari, Mehdi Salehi, Wanyun Catherine Gu, and Jeremy Clark. Sok: Oracles from the ground truth to market manipulation. In *ACM AFT*, page 127–141, 2021.

[22] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *The Journal of ACM*, 32(2):374–382, April 1985.

[23] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In *TCC*, pages 529–561, 2017.

[24] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *Transactions on Programming Language and Systems*, 4(3):382–401, July 1982.

[25] Hammurabi Mendes and Maurice Herlihy. Multidimensional approximate agreement in byzantine asynchronous systems. In *ACM STOC*, pages 391–400, 2013.

[26] Atsuki Momose and Ling Ren. Multi-threshold byzantine fault tolerance. In *ACM CCS*, pages 1686–1699, 2021.

[27] N.F. Neves, M. Correia, and P. Verissimo. Solving vector consensus with a wormhole. *IEEE Transactions on Parallel and Distributed Systems*, 16(12):1120–1131, 2005.

[28] Amirmohammad Pasdar, Young Choon Lee, and Zhongli Dong. Connect api with blockchain: A survey on blockchain oracle implementation. *ACM Comput. Surv.*, oct 2022.

[29] Amirmohammad Pasdar, Young Choon Lee, and Zhongli Dong. Connect api with blockchain: A survey on blockchain oracle implementation. *ACM Computing Surveys*, September 2022.

[30] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[31] Pyth Data Association. Pyth network: A first-party financial oracle. Technical report, Pyth Data Association, 2022. [Online: accessed on 21-Jan-2023].

[32] Hubert Ritzdorf, Karl Wüst, Arthur Gervais, Guillaume Felley, and Srdjan Capkun. TLS-N: non-repudiation over TLS enablign ubiquitous content signing. In *NDSS*, 2018.

[33] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.

[34] Lav Varshney. `https://slate.com/technology/2019/10/round-floor-software-errors-stock-market-battlefie` `html`, October 2019. [Online: accessed on 16-Nov-2022].

[35] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *ACMCCS*, pages 270–282, 2016.

[36] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: liberating web data using decentralized oracles for TLS. In *ACMCCS*, pages 1919–1938, 2020.