Empirical Measurements on Pricing Oracles and Decentralized Governance for Stablecoins

Wanyun Gu, Stanford University
Anika Raghuvanshi, Stanford University
Dan Boneh, Stanford University

Stablecoins are designed to address the volatility of crypto assets by maintaining a peg to a non-volatile currency such as the US Dollar. Decentralized Stablecoins that maintain their collateral on-chain need a pricing oracle to determine the current market value of the collateral. They also employ a decentralized governance system to make policy changes. In this paper we analyze the inner-workings of the pricing oracle and the decentralized governance mechanism employed in the MakerDao stablecoin, the largest and most popular on-chain stablecoin. We study the accuracy of the pricing oracle, as well as disagreements between pricing reports received by MakerDao. We also study the robustness of the decentralized governance system. This work sheds light on the practical operation of a pricing oracle and a decentralized governance mechanism in a large deployed system. We make a number of recommendations for improvements based on our findings.

1. INTRODUCTION

There are estimated 200 stablecoin projects in 2019 [4]. Stablecoins are crypto-native assets that are stable in value. Contrary to the well-known volatile nature of Bitcoin and Ethereum, stablecoins maintain a stable rate of exchange, usually in relation to some fiat currency such as the US Dollars. There are broadly speaking two types of stablecoins: ones that are collateralized *off-chain* by fiat currencies, such as Tether (USDT), and ones that are collateralized *on-chain* by crypto assets, such as MakerDao. An algorithm governing a stablecoin achieves stability by regulating the supply of stablecoin through buying and selling the stablecoins against the collateral assets.

A key component in the design of a stablecoin is a pricing oracle that aggregates data from external sources and posts them on the blockchain. The goal of the pricing oracle is to approximate the market value of the underlying collateral assets as accurately as possible in real-time. It is common practice for the system to select a set of price reporting feeds, which may be crypto exchanges, over-the-counter market makers, or traders, and task them with providing price update of the corresponding collateral asset at regular intervals. Designing an incentive-compatible mechanism to ensure truthful price reporting is critical to the long-term value of the network.

A poorly designed price-reporting system can be exploited by malicious attackers. Indeed on June 25, 2019, there was an oracle attack incident on the Synthetix Network Tokens (SNX). The system had only two commercial APIs at the time serving as price feeds for its forex product, sKRW, and one of the APIs started to intermittently report a corrupted price that was inflated by 1000x [22]. The incident lasted an hour and because the Synthetix protocol computes the average value of the feed reports and sends the output to the exchange rate contract, the contract for sKRW reported a corrupted reference price from which a sophisticated trading bot profited approximately $37 million from the price arbitrage [20].

Another important component of a stablecoin is a governance system which is used to update internal stablecoin policies. If the governance system is not properly decentralized, adversarial entities could use their voting power to create a 51% attack and hijack the system. In the context of stablecoins, an attack like this could involve depegging the currency to create a forged arbitrage opportunity, for example, by accepting an invalid price report from an untrusted data source. It could also result in theft of part of the collateral maintained on-chain by the system. A governance attack will have significant implication on the security of the protocol overall, and may directly impact the stability of the stable token. Designing a robust incentive-compatible governance scheme can help mitigate these risks.

In this paper, we examine the pricing oracle and the governance system in the MakerDao protocol, which facilitates one of the largest and most popular decentralized lending platforms to date [6]. The Maker team launched its stablecoin Dai, an ERC20 token, initially as an on-chain single-collateral stablecoin at the end of 2017. In the original V1 version, Dai is collateralized solely by Ethereum (ETH) and is supported by an oracle system that aggregates ETH-USD pricing data from 14 sources. Since November 2019, MakerDao has expanded the network to a multi-collateral system and to a larger set of oracle data sources.

Summary of our results. We begin in Section 2 with a survey of the oracle governance introduced by a number of stablecoin and decentralized projects, focusing in more detail on MakerDao. In Section 3, using historical data based on the single-collateral system, we present our analysis of the accuracy and reliability of the MakerDao pricing oracle. We show that at certain times the oracle can deviate from the true ETH-USD price by as much as 17%. Similarly, there can be disagreements between the data sources, and there are periods when the oracle uses mostly stale data to compute its price estimate. In Section 4 we look at alternative designs for the oracle aggregation algorithm, and compare their performance to MakerDao's algorithm. We also provide a statistical approach for detecting misbehaving oracles. Finally, in Section 5 we examine the level of participation and robustness of MakerDao's governance system. We find that due to low voter participation, a small number of MKR owners can sway the result of an election in their favor. Overall, while the MakerDao oracle and governance systems are well designed, we make a few recommendations for improvements, in particular to the governance system.

2. A REVIEW OF PRICING ORACLES

An oracle is a mechanism to aggregate and broadcast data from external off-chain sources onto the blockchain. For a stablecoin, the oracle is used to approximate the fair value of the underlying collateral asset in order to guarantee collateralization and thereby the stability of the stablecoin through time. Below we review a number of stablecoin projects with differing oracle reporting mechanisms. We also examine a few notable pricing oracle designs used in other decentralized applications. We then dive more deeply into MakerDao.

Reserve Protocol. A decentralized stablecoin system with on-chain collateral backing that has not yet launched. In its white paper [2], the protocol describes The Market Feed for which a collection of off-chain, trusted Reporters are chosen to periodically fetch market data that summarizes the last 30 minutes of trading from each major exchange on which an asset trades, including information on volume, volatility, and average price, and sends the results back to the on-chain Record Book. The protocol then calculates a volume-and-time weighted average of prices across trades, known as the 30-minute fast average. The Reserve team states in their white paper that at launch, it plans to rely on reporters that the Reserve team has built itself, and only over time will the protocol shift to a decentralized oracle.

Terra. A stablecoin project designed for decentralized finance launched in April 2019. The protocol [11] relies on the miners to report and validate the oracle prices through a weighted median voting mechanism scheme. Miners submit a vote for what they believe is the current exchange rate of the collateral asset, and for a sequence of n blocks, for some n, the vote is tallied by taking the weighted median as the true rate. The mechanism rewards those whose votes are within one standard deviation of the elected median and slashes the stakes of those whose votes are outside the range. Staking schemes such as this may encourage large miners to collude and rig the price. The Terra protocol reduces the potential threat by setting time-locks to the reward: since miners are a subset of users with strong vested interest in the system, misbehavior will damage the system, but the potential rewards gained from cheating will be reduced because of the time-locks.

SchellingCoin. A proposal by Vitalik Buterin for creating a minimal-trusted universal data feed [3]. For the consensus protocol in SchellingCoin, truth reporting should be the most powerful consensus, known as the *Schelling point*. The concept was first pioneered in 1960 by Thomas Schelling, a

Nobel laureate in Economics, in his seminal book *The Strategy of Conflict*. It describes coordinating behavior in game theory for players to converge on some focal point in the absence of communication. The SchellingCoin protocol describes a cryptographic scheme for which all users can act as price feeders. In each block, every user submits a cryptographic commitment to an individually reported price along with that user's address. In the next block, users open their commitments and submit the committed prices from the last round. Users whose submitted values lie between the 25^{th} and 75^{th} percentile are rewarded according to their prices. The consensus price should be close to the true price because the best strategy for each user is to report the correct value, given the incentive structure. However, the protocol is vulnerable to collusion attacks, where a large enough coalition might deviate from the Schelling point. A study by Crawford et al. [5] suggests that this method may fail to obtain the correct price in case of asymmetric payoffs. For example, bribery or a credible *promise* of bribery, may cause users to deviate from an honest strategy and break the Schelling point [10].

Augur. A decentralized oracle and a platform for prediction markets launched in August 2018. The Augur oracle [21] uses the idea of Schelling point, and specifies a price reporting mechanism and a dispute mechanism. For any prediction market that has been created, the market creator stakes and designates a reporter to submit a report within a pre-defined time window. The reporter is incentivized to report the truthful outcome: it is rewarded if the report is consistent with consensus, and penalized otherwise. During the subsequent dispute round, any holders of Augur's native token REP can challenge the report by staking REP holdings. Successful disputes on false outcomes are rewarded with a fixed return; unsuccessful disputes will return the stakes to the original owners.

ChainLink. A decentralized oracle network launched in May 2019. The protocol combines an on-chain architecture that is composed of three contracts [7] — reputation contract, order-matching contract, and aggregating contract — with off-chain data reporting mechanism. The on-chain reputation contract tracks the performances of all oracle service providers. A requester proposes a new service level agreement (SLA) who can choose a preliminary set of potential oracle providers based on their reputation. The SLA is submitted to an order-matching contract for the oracle providers to bid on the proposal. Bidding requires the providers to be committed, by staking an amount that will be slashed in the case of misbehavior. Once the set of oracle providers have been selected, they execute the SLA off-chain and report back the outcomes on-chain through the aggregating contract, which computes the final output and updates the reputation metrics. However, the protocol does not specify any particular aggregation method, because depending on the type of data input, the protocol would like to customize the aggregation method accordingly. Thus, methods including majority voting or weighted averaging can all be applied on-chain. In the medium to long run [7], ChainLink would like to migrate the computation step of aggregation off-chain to reduce cost and latency. However, this poses potential threat where oracle providers can communicate off-chain, and the blockchain will lose direct visibility to monitor their responses.

UMA. A platform for decentralized financial contracts that has not yet been launched. The protocol describes a Data Verification Mechanism (DVM) [12] that focuses on verifying the accuracy of off-chain price reports being brought on-chain. Similar to Augur, UMA resolves disputes through a Schelling point voting system where voters with tradable voting tokens can participate. Those who voted on the correct dispute outcome are rewarded while others are penalized. For an adversary to manipulate the voting outcome, it must control some minimum threshold of votes. Given the unit price of each vote — a variable that can be controlled by the protocol — the Cost of Corruption (CoC) can be calculated. UMA also requires that all contracts to be registered with the DVM, recording the monetary worth of each contract if an outcome was to be realized. The total value of all contracts in the system reflects the Profit from Corruption (PfC). Thus, if the cost of an attack (CoC) falls, the protocol will levy a fee on all contracts pro rata to purchase and burn voting tokens and push the price of vote up, ensuring that CoC $>$ PfC, to maintain system security.

```
function compute() returns (value, boolean):
   list = []
   for (value, valid_until) in price_feeds:  // for each price feed contract,
       if now < valid_until:                 // if the value is not expired,
          list.append(value)                 // append value to list.

   if len(list) < min:                       // if list is too short (default min = 1)
       return (last_median, False)           // return old value

   median = list.median()                    // compute the median value
   return (median, True)
```

Table I: MakerDao V1 Medianizer algorithm

2.1. MakerDao V1

MakerDao V1 was launched at the end of 2017. MakerDao is a collection of contracts on the Ethereum blockchain, with the goal of providing decentralized financing on-chain. Dai is Maker's stablecoin. To mint new Dai, in the initial V1, individuals must open Collateral Debt Positions (CDPs) to deposit and lock away Ethereum (ETH), and receive Dai in return. There is a minimum required collateralization ratio that the CDPs must maintain in order to avoid liquidation. The ratio is currently set at 150%, i.e. a CDP position with $100 Dai minted requires a minimum of $150 worth of ETH deposited. Given that the collateral asset for Dai is ETH, the Maker oracle needs to update the ETH-USD exchange rate in real-time to properly enforce the collateralization ratio. The ETH-USD exchange rate is known as the reference price, and the Maker's Medianizer smart contract [16] computes this price.

Maker has whitelisted 14 price feed contracts who can post price updates to the oracle. The source of each price feed is anonymous, but each feed is chosen as independent operator to "monitor the reference price across a number of external sources". However, neither the contract nor MakerDao's blog posts have clarified the *type* of price feeds that are collected, for instance whether the feeds are the mid, the minimum, or the maximum price observed in any given hour. The price feeds also do not provide transactional volume information.

The Medianizer algorithm. To update the ETH-USD price, a whitelisted price feed contract calls the `post` function on the `PriceFeed` contract with arguments `value, valid_until, Medianizer_addr`, where:

— `value` is the claimed current ETH-USD price;
— `valid_until` is a datetime that indicates when the posted value expires;
— `Medianizer_addr` is the contract address of the Medianizer.

Every call to `post()` results in a call to the Medianizer's `compute()` function to update the ETH-USD reference price. The `compute()` function aggregates all currently *valid* posted values from the 14 price feeds, and computes their median. In other words, a value submitted by a price feed contract will be used in the median calculation until time `valid_until`. We call this method *forward fill*, since if a price feed has not posted a new price, the Medianizer will use the older price to fill the value for the current time, as long as the previously posted price has not expired. The complete algorithm used to compute the ETH-USD price is shown in Table I.

The Maker Feeds website states that the oracle sets a limit of 6 hours on `valid_until`. However this limit is not enforced in code. As we will see, the six-hour limit has not been followed by several price feeds who post values with a `valid_until` that is 12 hours in the future.

2.2. MakerDao V2

Maker recently announced the V2 multi-collateral system and it added more sources of price feeds [19]. The additions include a set of partners, whose public identities are disclosed: *0x, dYdX, Set Protocol*, and *Gnosis*. The Maker team also introduced a new Oracle Module logic, designed to better incentivize active and honest price reporting behavior. An Oracle Module is built for each collateral asset, which contains two core components: the Median contract and the Oracle Security

Module (OSM) contract [17]. Whitelisted price feeders can broadcast price updates off-chain, which are then aggregated on-chain by calling the Median contract. The median price value is then pulled into the OSM contract. The logic for processing price feeds is primarily in the Median contract. We describe below how it works, highlighting some of the new features.

Median contract. To update the reference price of any given collateral asset, a whitelist of price feeders are chosen by governance through permissioning logic. To update the reference price for a collateral asset (e.g. ETH), one calls the `poke` function on the Median contract [15] with arguments called `val_`, `age_`, and `(v, r, s)`, where:

— `val_` is a sorted array of price feeds, one entry from each price feeder;
— `age_` is an array of ages for each price feed in the `val_` array (the age of a price feed is the block timestamp at the time that the reference price for the asset was measured);
— `(v, r, s)` is an array of signatures: entry number *i* in this array is a signature by price feeder number *i* on the triple (`val_`[*i*], `age_`[*i*], AssetType), where AssetType is a fixed string such as `'ETHUSD'`.

The `poke` function attempts to update the reference price of the collateral asset. It uses an internal value `bar` that is the size of the quorum required for the median price to be updated. Thus, a key feature in the V2 Median contract is that the median is only updated if there are at least `bar` unique signed price feeds since the last median update. The value of `bar` is set through governance. It is currently set at thirteen for the two collateral assets, Ethereum (ETH-USD) and Basic Attention Token (BAT-USD), available on the Maker platform [8; 9].

The `poke` function rejects the submission, unless the following conditions hold:

— the length of the `val_` array is exactly `bar`, i.e. the contract currently requires `bar` price reports, no more and no less;
— the `val_` array is sorted in ascending order and contains only positive values;
— for each price report, the contract requires that:
  — the signature `(v,r,s)` on the corresponding value and age inputs is a valid signature;
  — the signer is permissioned (i.e., only price reports from authorized reporters are accepted);
  — the signer signed only one entry in the provided array (i.e., all thirteen price reports are from different signers);
  — the age of the price report is less than the age of the last median update (no stale pricing);

If all the checks above pass, `poke` sets the updated asset price to be the median of the reported prices, which is the middle value of the `val_` array. It also records the current block timestamp.

While the Median contract tries to prevent stale pricing by requiring `bar`-number of prices to be updated in order to compute the latest median, unlike Augur or UMA, there is no direct incentive scheme for individual price feeders beyond the Oracle Module. Interestingly, if more than `bar` fresh price reports are available, the current implementation will not fully utilize all the information. It requires the caller to choose a subset of exactly `bar` of them, and ignores the remaining fresh reports.

The MakerDao V2 Oracle Module was only created recently, and there are currently *circa* 1000 on-chain transactions issued to it [8; 9]. There is considerably more data for the MakerDao V1 oracle contract. Therefore, in the next section we will primarily analyze the price feeder data sent to the MakerDao V1 contract.

3. A HISTORICAL VIEW OF MAKERDAO'S ORACLE AND PRICE FEEDS

Dai was officially launched on Dec 18, 2017. We downloaded event logs and historical external price feeds posted to the Medianizer contract by the whitelisted addresses for the period of Oct 17, 2017 to May 17, 2019, by calling *The Etherscan Ethereum Developer APIs*.

Using the historical price feed data, we examine three types of anomalies:

— **External disagreement:** time slices when there is a disagreement between the price feeds and a publicly reported ETH-USD reference price,

— **Internal disagreement:** time slices when there is a large deviation between the reported price feeds from the 14 price feeders, and
— **Stale data:** time slices when the reference price is computed from data that is several hours old, potentially causing the value to deviate from the truth. We also look at times when many price feeds are expired, making it easier for a small number of feeders to manipulate the price.

We are interested in the magnitude of these anomalies and their frequency.

The fact that internal and external disagreements exist is not too surprising. Truthful price feeders may report extreme values from time to time due to idiosyncratic conditions, such as local market liquidity, the size of the order book, and transaction costs. Unlike Terra, Maker does not punish feeders who report extreme values. Nevertheless, understanding the magnitude and the frequency of these events is informative.

3.1. External disagreements

We begin by examining points in time when the price feeds disagree with a publicly reported ETH-USD reference price.

The CCCAGG index. As our benchmark for the ETH-USD reference price, we use a volume-weighted average index from `CryptoCompare`, an independent platform that maintains the Crypto Coin Comparison Aggregated Index (CCCAGG) for a list of cryptocurrencies. CCCAGG aggregates transaction data from over 70 exchanges as price feeds to compute the volume-weighted averages. We chose the CCCAGG index because it contains a large number of price feeds and several dimensions of price information, including different price types (i.e. close, open, low, high) as well as volume information. We believe that CCCAGG index approximates a ground-truth value of the underlying asset with high accuracy.

Periods of external disagreement. Figure 1a shows a histogram of the absolute differences between the MakerDao Medianizer and the CCCAGG index. The histogram resolution is $10 and the graph is on a logarithmic scale.

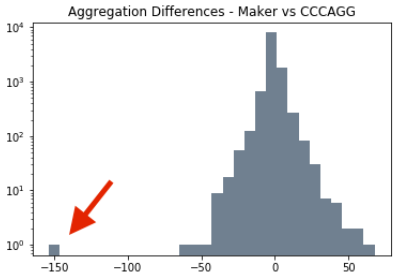
The figure shows a strong agreement between the MakerDao Medianizer and the CCCAGG index. For the majority of the hourly time slices the differences are small, varying on average by $3.33 with $4.79 standard deviation.

However, a few points stand out at the extreme ends of the histogram. On those occasions we observed an extreme price divergence between the MakerDao Medianizer and the CCCAGG benchmark. We examine three points in time that correspond to the largest deviations, and present the complete historical data for each of these events in Figure 1b. For completeness, the table also shows the age of every posted price from each of the 14 feeds at the time of the event. The age is the difference between the time of the event and the time that data was first posted by the feed. The age, rounded to an integer, is shown in parenthesis.

— The largest deviation occurred on 2018-02-02 05:00 where the price gap in ETH-USD was $153.73, representing a 17% divergence from the benchmark. At that time there were fourteen valid external price feeds, two of which were at least an hour old. All of them were quite far off from the CCCAGG benchmark.

— The second and the third largest divergences took place on 2018-01-10 19:00 and 2018-01-16 18:00, for which the price gaps were $67.89 and $59.84, respectively. Interestingly, nine out of fourteen valid price feeds collected for the second event were at least two hours old. Thus the price divergence from the benchmark could be the result of the oracle estimating the price based on stale data. It shows how the oracle can become inaccurate during a period of slow refresh and high market volatility.

For sanity check, we cross referenced the price of ETH-USD at all three times across several main crypto exchanges including Gemini and Coinbase, by calculating the mid of the high and the low



(a) Differences between the MakerDao Medianizer and CCCAGG. The x-axis is the price difference in USD. The y-axis is the number of times this difference was observed.

	2018-02-02 05:00		**2018-01-10 19:00**		**2018-01-16 18:00**	
Feeder 1	748.19		1314.00	(2)	1066.64	
Feeder 2	748.19		1315.00	(2)	1066.64	
Feeder 3	824.02		1309.22	(2)	1063.69	
Feeder 4	742.00		1288.00	(1)	1066.64	
Feeder 5	825.31		1308.40	(2)	1058.90	
Feeder 6	794.78		1306.60		1052.00	
Feeder 7	754.90		1307.06	(2)	911.13	(4)
Feeder 8	755.15		1299.00	(1)	1074.29	
Feeder 9	755.98		1306.60		1055.00	
Feeder 10	754.88		1308.90	(2)	1066.61	
Feeder 11	739.93		1311.99	(2)	1064.00	
Feeder 12	780.00	(1)	1295.01	(1)	1061.66	
Feeder 13	749.36		1308.00	(2)	1070.00	
Feeder 14	876.30	(3)	1311.98	(2)	1063.50	
Maker Medianizer	**755.02**		**1308.20**		**1063.85**	
CCCAGG	**908.75**		**1240.31**		**1004.00**	
Coinbase	930.80		1289.51		1039.00	
Gemini	935.62		1288.78		1039.86	

(b) Price feed data in dollars per ETH for extreme difference points. The age of each feed, rounded to the nearest hour, is shown in parenthesis when the age is non-zero.

Fig. 1: Differences between the MakerDao Medianizer and the CCCAGG index.

prices observed on each of the exchanges for the given hour. The price quotes on these exchanges were consistent with the price computed by the CCCAGG index, as shown in Table 1b.

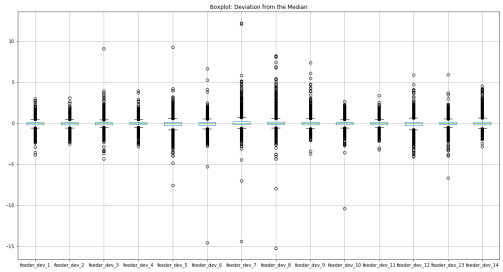
3.2. Internal disagreements

Next, we look at large disagreements between the 14 price feed contracts. We computed the median price of ETH-USD at each hour using only the fresh price feeds posted at that hour. We define deviation for each price feeder at time t as:

$$feeder\_dev(t) = \frac{price\_feed(t) - median(t)}{median(t)} \times 100. \tag{1}$$

Figure 2a shows the distribution of each feeder's deviation from the median through time. The y-axis shows the values obtained from (1), computed for each feeder for all time values t in our dataset. The deviations from the median for all feeders are close to zero. This is expected if all feeders behave truthfully most of the time and there is no systematic deviation by any single feeder.

Nevertheless, Feeders 7 and 8 exhibit the largest deviations from the median in price reporting. The standard deviation measuring their price deviations from the median is 0.75% and 0.71%, re-



(a) Feeder deviations from the median for all 14 data sources. The y-axis values are computed from (1).

Time Slice	feeder_8
2017-12-22 08:00	−15.23%
2017-12-12 19:00	8.27%
2018-01-16 06:00	8.17%
2017-12-19 17:00	8.13%
2018-01-16 04:00	−7.93%

(b) Periods of large deviation for Feeder 8 as computed from (1)

Fig. 2: Measuring large deviations among price feeders

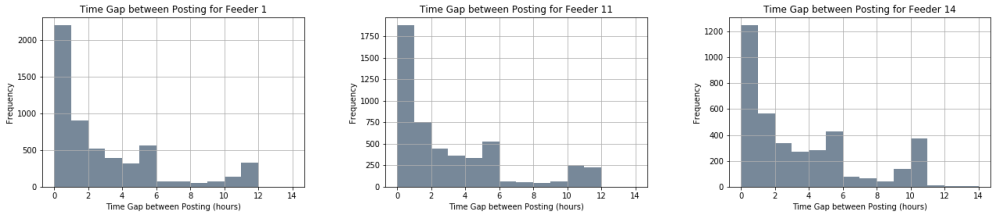


Fig. 3: Gaps between consecutive posts from Feeders 1, 11, and 14, respectively. The y-axis is the prevalence of a particular hourly gap.

spectively. One may wonder whether the behavior of Feeders 7 and 8 is due to random fluctuations in price, or whether there is something different about how they compute the ETH-USD reference price. We will come back to this question in Section 4.1.

The largest ever price deviation from the median by absolute percentage terms is committed by Feeder 8. In Table 2b, we list the top five largest deviation events observed for Feeder 8. It demonstrates that the price values posted by Feeder 8 can be quite far from the computed median. Similarly the largest deviation from the median by Feeder 7 was −14.36% which is also substantial. Fortunately, these deviations only negligibly impacted the value of the median.

3.3. Measuring stale and expired data

Finally, we look at the frequency at which the price feeders provide fresh ETH-USD prices. Clearly, pricing data that is several hours old can lead to an incorrect oracle for the current ETH-USD price. Moreover, if data from a price feeder expires, that data is no longer used in the median calculation. If the number of active price feeds is small, say two or three, it becomes quite easy to manipulate the ETH-USD price estimate, as happened in the case of SNX mentioned in the introduction. In this section we look at data freshness and the number of active feeders in the historical record.

Looking at the number of price feeds posted by each of the 14 price feeders shows that they all contributed about the same number of price reports over the measured time period. Feeders 6 and 8 were the most prolific, posting over 8000 prices, while Feeder 14 was the least, posting under 4000 prices. Interestingly, Feeder 8 was also the least accurate, as shown in Figure 1a.

The main question we are interested in posing is how frequent did the Medianizer contract use stale data from the data feeders in computing the reference price. Figure 3 shows the time gap between consecutive postings for three feeders, Feeders 1, 11, and 14, chosen arbitrarily among the fourteen price feeders. The figure shows that there are about 500 time periods where the time gap from these feeders was over 10 hours old. The same is true for several of the other price feeders.

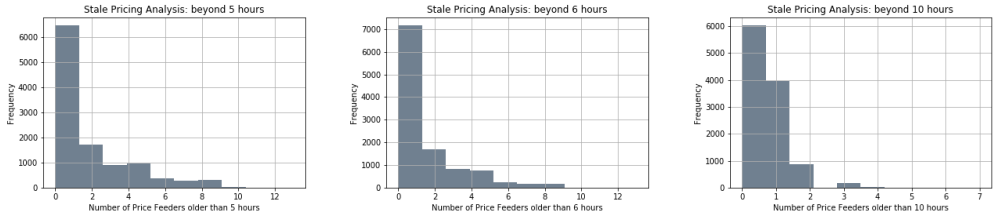


Fig. 4: The number of times x feeders were simultaneously stale by more than 5, 6, 10 hours, respectively.

Now that we see that the data for the price feeders can be several hours old, we next ask whether that can happen simultaneously across multiple price feeders. If so, then the estimated ETH-USD price computed by the Medianizer contract is effectively based on stale data, and is likely to be off in case of market volatility.

Figure 4 examines simultaneous staleness. For example, the right-most chart in Figure 4 shows that there are many (about 100) time periods when the data from **three** price feeders is over **ten** hours old. The middle chart shows that there are many (about 100) time periods when the data from **eight** price feeders is over **six** hours old. The left most chart shows that there are almost 200 time periods when data from **eight** price feeders is over **five** hours old. Concretely, Figure 4 demonstrates that there are many time periods where the majority of price feeder data is over six hours old, and is included in the median calculation.

How to handle stale data is a problem that all oracle-based systems must contend with. In the next section we look into whether Maker could have used a different method for calculating the ETH-USD estimate that would have handled stale data better.

4. COMPARING DIFFERENT MEDIANIZER ALGORITHMS

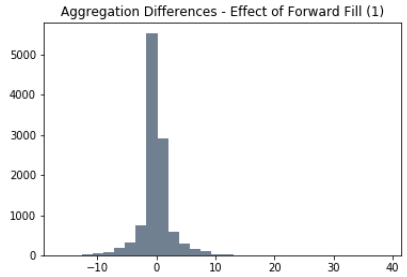
The historical analysis in the previous section shows that aggregating the 14 price feeds into a single ETH-USD value can be subtle, especially in periods when there are few active price feeders, or at times when the prices they contribute exhibit large variance or differ from the ground truth. In this section we ask whether the current Medianizer algorithm — median with forward fill subject to expiration criteria — is the most accurate method to aggregate the price feeds into a single value. The concern is that the forward fill algorithm may cause the estimated ETH-USD price to be based on stale data that is several hours old.

To explore the accuracy of the forward fill method, we experiment with different ways of aggregating the data into a single ETH-USD price, and investigate the accuracy of each method, compared to the current MakerDao algorithm.

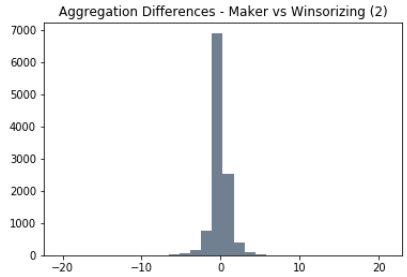
Alternate Medianizer strategies. We compare MakerDao's algorithm with two alternate methods.

— **Median without forward fill.** Here we compute the ETH-USD reference price using only values that are posted by the price feed contracts within the current hour.
— **Winsorized mean.** Here we first form the hourly dataset by replicating Maker's Medianizer approach, by forward filling prices if they are valid at the time the oracle snapshot was taken. However, instead of computing the median, we calculate the truncated mean of the values. Specifically, for a sorted list of non-expired price feeds, we mask the 5% lowest values by replacing their values with the value corresponding to the 5th percentile, and we mask the 5% highest values by replacing with the value corresponding to the 95^{th} percentile. The result is called a winsorized dataset, and we take its mean as the ETH-USD price.

Figure 5 shows the absolute difference in price between the MakerDao median and each aggregation method. Figure 5a measures the impact of forward fill. For the entire price history, 90% of the Aggregation Differences is non-zero implying that the computed median values differ in value as a



(a) Difference between the median with forward fill and without



(b) Difference between the median with forward fill and the Winsorized mean

Fig. 5: Evaluating alternate aggregation methods

result of forward fill. However, the average absolute difference is small with a mean of $1.43. The histogram shows that differences in computed median values due to the effect of forward fill follows a Gaussian distribution with small kurtosis, suggesting there is little difference in calculating the median by either methodology.

Figure 5b shows that the difference between MakerDao and the winsorizing mean is also small. While the winsorizing mean is closer to the median with forward fill than without, the differences are small.

The conclusion is that when using the current set of price feeds, all three methods produce very similar results. Hence, data staleness in forward fill is not a significant concern. Moreover, the forward fill method has a robustness advantage over the non-forward-fill method. The reason for this is that without forward fill, the median is computed over a smaller set of values, and is therefore more vulnerable to manipulation. Since accuracy does not seem to be impacted much by forward fill, the robustness benefits of forward fill justify Maker's choice.

For MakerDao V2, the `Median` contract introduces a quorum parameter `bar` in order to update the median (Section 2.2). This is designed to address the potential problem of stale pricing to prevent malicious attacker from manipulating the median price output. However, beyond the Oracle Module the protocol does not currently contain any direct incentive schemes for individual price feeders to provide accurate reports. As the platform expands, the Maker team can consider adopting some Schelling-style voting system that helps to reinforce both the frequency and the quality of the price feeds.

4.1. A Statistical Test for Truthful Reporting

In this section we develop a simple statistical test to help the Medianizer contract detect an incorrect price report from a data feeder. The approach works by following these steps.

(1) For each of the 14 data feeds we compute a time series of feeder deviations as defined by Equation (1). We obtain 14 time series. By plotting each time series we confirmed that they all closely follow a normal distribution with mean close to zero. However, each time series has a different standard deviation.
(2) We compute the maximum-likelihood mean and standard deviation for each of the 14 time series. This gives us the normal distribution parameters for each of the 14 time series.
(3) Finally, for every price posting from a data feeder we compute the p-value for that posting using the computed parameters in step (2). If the p-value is less than 0.05, we declare the posting as invalid.

We test this approach on the large internal disagreement events for Feeder 8 reported in Figure 2b. Recall that on `2017-12-22 08:00` the price reported by Feeder 8 had a price deviation from the median of 15.23%. Computing the p-value for this based on Feeder 8's mean and standard deviation of the sampled price deviations from the median over time, which are 0.09% and 0.71%, respectively, gives a p-value of approximately **0.00**. This indicates that given Feeder 8's past accuracy, this size deviation is very unlikely, and the data provided will be ignored. In the historical record, this test would cause 3.8% of the price feeds from Feeder 8 to be ignored.

5. GOVERNANCE ATTACKS

We now turn to a different aspect of the system, and examine the impact of an attacker who attempts to exploit the governance structure. We begin by describing the current state of MakerDao's governance mechanism, and then analyze the level of centralization in the voting system.

5.1. Voting in MakerDao

In addition to the Dai coin, Maker has a second coin for governance called MKR, which the governance participants use to vote. The amount of MKR a voter owns directly correlates to their voting power, so we treat 1 MKR as equivalent to one vote. MKR holders cast their votes by locking up their MKR into the voting contract. There are two categories of votes: Governance Polls, which aim to find a resolution to a matter, and Executive Votes, which change the state of the system:

— "Signal your support to set the Stability Fee to 20%" is a Governance Poll
— "Change the Stability Fee to 20%" is an Executive Vote.

For Governance Polls, several proposals may be suggested and the one that receives the most votes, even if this is not a majority vote, will be accepted. For an Executive Vote to pass, it must get more votes than the previous Executive Vote that passed. Executive Votes follow a Continuous Voting process: votes have no time limit and never 'expire'. So a new proposal will be executed when enough voters decide to transfer their votes from the old proposal to the new. Neither type of vote has a minimum voter turnout. Governance Polls usually last two to three days while Executive Votes have no time limit due to continuous voting [14].

To become a participant in the MakerDao governance system, a potential voter must first set up a voting contract to lock up some amount of MKR, which can then be used to vote. Setting up the voting contract involves connecting a cold wallet (Ledger or Trezor) that contains MKR and ETH and a hot wallet with ETH. Only a small amount of ETH is needed in both the hot and the cold wallets for gas costs since setting up the voting contract and voting itself are on-chain transactions. An MKR voter can reuse its token by withdrawing MKR from old votes, and casting it towards new ones. The process to cast and withdraw votes can be completed on the voting dashboard. This is straightforward provided that the voter has enough ETH to pay for gas costs. Due to the Voting Proxy contract [18], voters can use their full MKR value for both Governance Polls and Executive Votes simultaneously.

5.2. Current Centralization Trends

There is exactly one million MKR in existence. As described above, the Continuous Voting mechanism makes it such that a new Executive Votes will execute once a majority of MKR holders have transferred their MKR to the new vote. For example, if an old proposal had 100 votes, a new proposal could execute once 51 of these votes were transferred to it. Or alternatively, even if no votes were transferred, the new proposal could win if additional voters came in and cast 101 votes towards the new proposal. The most recent Executive Vote was executed when voters put 45,080 MKR towards it, meaning that this was the turning point at which the new proposal had more votes than the old proposal. Given this, we can extrapolate that about 90,000 MKR is actively being put towards Executive Votes.

Now, let us consider what would happen if a malicious entity wanted to pass an Executive Vote. We consider the situation which would make an attack most difficult: although the last Executive Vote was executed with 45,080 MKR, after some time voters will likely have transferred most of

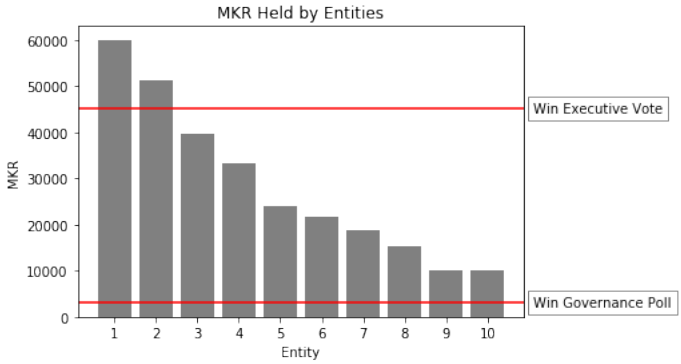


Fig. 6: Amount of MKR held by individual entities as of January 31, 2020.

their MKR over to the new proposal. Suppose the current proposal has 90,000 MKR. If a malicious entity wanted to pass a new Executive Vote, it would have to put more than 90,000 MKR to ensure that the new proposal is executed. With the current MKR price of *circa* $540, the cost of such an attack is $48.6 million. Figure 6 shows the breakdown of the top ten MKR holding entities. Raw data is collected from Etherscan and a tool called Bloxy [1]. Known exchanges are not displayed in this list, and to our knowledge, each of these holders are individual entities. The largest MKR holders individually own 60,000 and 51,291 MKR, respectively, thus they could collude to mount the attack described above. Additionally, if a new Executive Vote was passed immediately after a previous one that only had 45,000 votes put towards it, then each of the two entities could single-handedly mount the attack. All the other eight largest MKR Holders also individually hold over 10,000 MKR. These individual entities can overpower the majority of voters through collusion.

Each Governance Poll contains voting options and the option with the most votes will win. Table II shows voter turnout for the most recent batch of Governance Polls that were held between January 27 and 30, 2020. Our raw data is gathered from MakerDao's governance dashboard [13]. We see that Governance Polls do not have a huge amount of participation. These polls all had less than 3% turnout in terms of the number of 'votes' (MKR) they received. Out of the four Governance Polls ending on January 30, 2020, the one with the most votes received 29,908 MKR out of the one million MKR in existence. Furthermore, the amount of MKR cast for the winning vote is an even smaller percent, and a couple of the most recent governance polls won with less than 4,000 MKR. In Figure 6, we show a horizontal red line at the number of votes that won one of the most recent Governance Polls: the Sai Stability Fee Adjustment on Jan 27, 2020. We see that the amount of MKR used to execute this vote is far below all top ten MKR holding entities. We additionally notice that out of the current 14,718 unique MKR holders [1], only about twenty of them voted on each of the Governance Polls, translating to less than 1% for each poll.

We briefly mention some risks that could hypothetically occur if a malicious party successfully creates a centralized governance attack.

— **Choice of price feeders.** The malicious party could elect a completely new set of external price feeds to compute the reference price of collateral asset, since it is calculated as the median price feed. Inside the MakerDao V2 median contract, for example, if a malicious entity can change the `bar` parameter that represents the size of the quorum, or if the attacker can successfully permission a new set of price feeds to replace the existing quorum, they could skew the value of the reference price. This would cause the Oracle to use an incorrect price P_{oracle} to determine the value of CDPs and thereby the supply of Dai in circulation, destabilizing the stablecoin from its par value to the dollar. Even if the market stabilized and the system recovered from this attack,

Poll Name	% of MKR Voted With	% MKR Used in Winning Option	% of Unique Voters who Participated
Dai Savings Rate Spread Adjustment	0.64	0.04	0.16
Dai Stability Fee Adjustment	1.59	0.10	0.15
Debt Ceiling Adjustment	2.99	2.99	0.15
Sai Stability Fee Adjustment	0.62	0.03	0.12

Table II: Voter turnout for Governance Polls executed on January 30, 2020.

during the period when Dai was de-pegged, the attacker can take profits through price arbitrage similar to the Synthetix oracle attack that had occurred.

— **Emergency oracle.** The malicious party could additionally elect a new Emergency Oracle, who has the power to unilaterally trigger Emergency Shutdown. Emergency Shutdown is embedded in MakerDao V2 Oracle Module. It is capable to shutdown the price feeds and halt median update, but is incapable of bringing the feeds back up without governance to take control [15]. During the shutdown period, this can create potential arbitrage opportunities to manipulate the platform, causing serious consequences to the network security.

5.3. Addressing Governance Centralization: the Way Forward

Despite the potential for governance attacks, progress is being made towards decentralizing holders of MKR. From the first Governance Poll in September 2018 to January 2020, there is an increase of unique MKR holders from 4,028 to 14,718 [1]. However, this does not mean all of those MKR holders have opened voting contracts or actively participate in votes.

The Maker system appears to have a few very committed voters, which keeps the system healthy. However, voter turnout has been shown to be an issue, as it puts the system at risk of attack. Here we briefly discuss three methods to further encourage participation in governance. We hope that these recommendations will be applicable to many stablecoins with on-chain governance.

Simplify the voting process. Currently, a party that wishes to participate in the MakerDao governance system must go through a long series of steps. This relatively high barrier to entry creates another point of centralization. Here we describe the steps presently required to prepare for and set up a voting contract. The following are the prerequisites for setting up a voting contract:

(1) A hot wallet with ETH funds.
(2) A Ledger or Trezor cold wallet ($100 USD) bought online and shipped to a physical address.
(3) ETH funds in the cold wallet. To transfer funds to the cold wallet, the voter must download the correct software and set up a wallet on the Ledger or Trezor device. They must then create a transaction which sends some ETH from a hot wallet to the address of the cold wallet.
(4) MKR funds in the cold wallet, which will be used to vote. Additional steps may have to be taken in the case that MKR is not offered on the voter's preferred cryptocurrency exchange. The MKR can be transferred to the cold wallet through the same mechanism as in step 3.

After these initial steps, the voter visits the MakerDao governance dashboard [13] to set up their voting contract. The dashboard will lead them through the process of connecting their cold and hot wallets and locking up the MKR that they will use to vote on proposals. Each step along the way – connecting hot and cold wallets, transferring funds from one wallet to another, and the creation of the voting contract – requires some small transaction fees paid in ETH.

Overall, the initial setup and somewhat involved process of creating a voting contract make it challenging for the average user to become a voter. Many of these steps can be automated, or hidden behind a user-friendly user interface.

Provide alerting tools. To promote accountability, the protocol team should provide tools to allow voters to easily monitor the state of decentralization. For example, the MKR Foundation controls a Dev Fund which contains enough MKR to win every Executive Vote. To limit the resulting central control of the system, the MKR Foundation states that addresses that are associated to the Dev Fund do not participate in the vote. However, this is not enforced in code, nor is there a tool that allows the users to easily verify. Adding a simple check in the code to block these addresses from voting will move the system further towards decentralization.

Another needed tool is to monitor the distribution of votes on a proposal. This could encourage people detect cases where one voter single-handedly swings the outcome with their voting power. Certain tools such as `Bloxy` and `Etherscan` provide some insight into these metrics retroactively, but on-chain alerting tools for users could prevent risky scenarios from occurring.

Allow votes delegation. Realistically we cannot expect a majority of voters to participate in all votes. Instead, the system could allow a MKR holder to delegate its votes to a proxy that they feel is similarly aligned. Currently, the Maker system does not support delegating votes to a proxy. Delegated votes may greatly increase voter participation and representation against any single malicious entity who might otherwise has the power to swing the outcome. We should, however, be wary of centralization in proxies. Clearly defined policies and verifiable actions through on-chain alerting tools can help mitigate this risk.

6. SUMMARY AND DIRECTIONS FOR FUTURE WORK

In this paper, we have examined decentralized pricing oracles using MakerDao as our empirical case study. Enforcing price feeds to provide reliable, fresh reports on-chain is a challenge that all oracle-based systems must contend with. We additionally explored potential paths for creating a fair and representative governance system. This work raises a number of directions for possible future work. First, following the introduction of Oracle Module logic in MakerDao V2, it would be worthwhile to conduct the same empirical measurements as we have done for V1 on the new Median smart contract, once there are enough data points for analysis. Moreover, it would be valuable to compare against other deployed systems that use pricing oracles, such as the Compound protocol. Second, it would be quite interesting to analyze the degree of centralization in other governance systems used by decentralized finance platforms.

Acknowledgments

This work was funded by NSF and the Stanford Center for Blockchain Research (CBR).

REFERENCES

1. Bloxy.inc. Maker Holders. https://bloxy.info/token_holders/0x9f8f72aa9304-c8b593d555f12ef6589cc3a579a2.
2. T. Brent, D. Colson, M. Elder, H. Fisher, N. Freeman, J. Ostman, and E. V. Nostrand. Reserve Stabilization Protocol. https://reserve.org/whitepaper.pdf, 2019.
3. V. Buterin. SchellingCoin: A Minimal-Trust Universal Data Feed. https://blog.ethereum.org/2014/03/28/schellingcoin-a-minimal-trust-universal-data-feed, 2014.
4. CementDAO. How Many Stablecoins Are There? https://medium.com/cementdao/how-many-stablecoins-are-there-aa39d201ac12, 2019.
5. V. P. Crawford, U. Gneezy, and Y. Rottenstreich. The Power of Focal Points Is Limited: Even Minute Payoff Asymmetry May Yield Large Coordination Failures. American Economic Review, Volume 98, Number 4, 2008.
6. DeFi Pulse. DeFi Pulse ranking Maker as the number one decentralized lending platform. https://defipulse.com.

7. S. Ellis, A. Juels, and S. Nazarov. ChainLink: A Decentralized Oracle Network. https://link.smartcontract.com/whitepaper, 2017.
8. Etherscan. Median Contract for BAT. https://etherscan.io/address/0x18b4633d6e39870f398597f3c1ba8c4a41294966.
9. Etherscan. Median Contract for ETH. https://etherscan.io/address/0x64de91f5a373cd4c28de3600cb34c7c6ce410c85.
10. J. Garay. On Oracles and Schelling Points. https://medium.com/witnet/on-oracles-and-schelling-points-2a1807c29b73, 2018.
11. E. Kereiakes, D. Kwon, M. D. Maggio, and N. Platias. Terra Money: Stability and Adoption. https://s3.ap-northeast-2.amazonaws.com/terra.money.home/static/Terra_White_paper.pdf?201904, 2019.
12. H. Lambur. UMA Data Verification Mechanism: Adding Economic Guarantees to Blockchain Oracles. https://github.com/UMAprotocol/whitepaper/blob/master/UMA-DVM-oracle-whitepaper.pdf, 2019.
13. MakerDao. MakerDao Governance Dashboard. https://vote.makerdao.com/.
14. MakerDao. MakerDao Governance Risk Framework (Part 3). https://medium.com/makerdao/makerdao-governance-risk-framework-part-3-7a4c620f4077.
15. MakerDao. Median: Detailed Documentation. https://docs.makerdao.com/smart-contract-modules/oracle-module/median-detailed-documentation.
16. MakerDao. Medianizer Smart Contract. https://makerdao.com/feeds.
17. MakerDao. Oracle Module. https://docs.makerdao.com/smart-contract-modules/oracle-module.
18. MakerDao. The MakerDAO Voting Proxy Contract. https://blog.makerdao.com/the-makerdao-voting-proxy-contract/.
19. MakerDao. Introducing Oracles V2 and DeFi Feeds. https://blog.makerdao.com/introducing-oracles-v2-and-defi-feeds, 2019.
20. J. OConnell. Sophisticated Trading Bot Exploits Synthetix Oracle, Funds Recovered. https://cointelegraph.com/news/sophisticated-trading-bot-exploits-synthetix-oracle-funds-recovered, 2019.
21. J. Peterson, J. Krug, M. Zoltu, A. K. Williams, and S. Alexander. Augur: a Decentralized Oracle and Prediction Market Platform v2.0. https://www.augur.net/whitepaper.pdf, 2019.
22. Synthetix. Synthetix Response to Oracle Incident. https://blog.synthetix.io/response-to-oracle-incident/, 2019.