A Data-Driven Primer on the Differences Between Oracles

Christopher Pondoc

July 9, 2021

1 Introduction

This project analyzes different oracles within the world of blockchain development through a data science lens. By observing and comparing various relevant metrics of oracles, I was able to gain some reasonable comprehension about the world of crypto more practically.

Note: To run a data app that collects the data in real time, visit: bit.ly/oracle-diff.

2 Background

Developing on the blockchain starts with smart contracts, which are self-operating computer programs, which live on the blockchain and execute when a certain amount of conditions are met [1]. Despite the many advantages of smart contracts, one limiting factor is the inability to get data from outside of the blockchain onto the blockchain. Such data can include information about the weather, random numbers, or price feeds of different currencies.

This is where oracles come in: these mechanisms help to retrieve data from off the blockchain for smart contracts to use for dispensing money, making decisions, and more [2]. In this specific report, we'll be looking at oracles and price feed data, and understanding how different oracles perform compared to one another.

2.1 Specific Oracles

In the case of this task, we'll be focusing on the following oracles. As a primer to the differences between the differences between these oracles and their implementations, check out the linked reference:

- Tellor [3]
- Chainlink [4]

- Band Protocol [5]
- DIA [6]

3 Process

The first step is to grab pertinent data from each of the oracles. In this case, I followed a specific process:

1. First, I found the address of each oracle's smart contract on the blockchain. For the most part, all the oracles provided the address to their smart contract through the documentation on their website.
2. After finding the addresses of each smart contract, I went to EtherScan, and downloaded each contract's Application Binary Interface, or ABI [7]. Since smart contracts are stored and compiled in the blockchain as bytecode, in order to communicate with a smart contract, we must use an ABI to determine which functions I can invoke as well as what format data will be returned to me [8].
3. Finally, using `web3.py`, I was able to connect each of the project's smart contract, invoke the proper arguments, and then calculate the metrics accordingly [9].

Below is a code sample that can be used as a template for connecting to a smart contract using `web3.py`. More code samples can be found in the `scripts` folder in the main repository [10]:

```
from web3 import Web3
import json

f = open('path/to/abi', 'r')
abi = json.load(f)
web3 = Web3(Web3.HTTPProvider('https://mainnet.infura.io/v3/PROJECT_URL'))
address = 'CONTRACT_ADDRESS'
f.close()
return web3.eth.contract(address=address, abi=abi)
```

4 Data Analysis

4.1 Price Feeds

The first data I looked at were simply the pure price feeds, mainly focusing on conversions between certain large cryptocurrencies.

Specifically, I decided to key in on the following conversions:

- BTC/USD

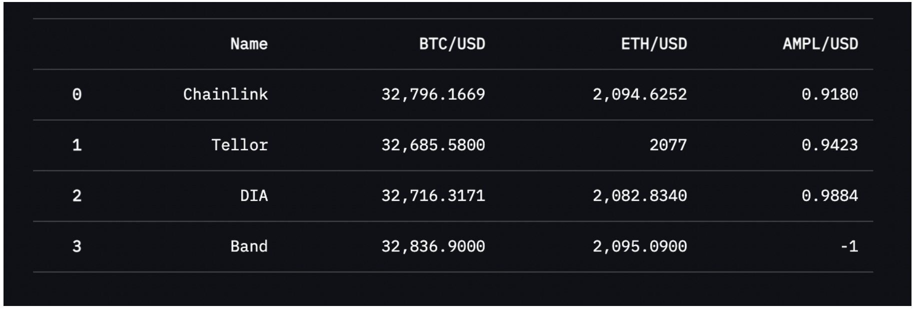
- ETH/USD
- AMPL/USD

For each oracle, I utilizes the below functions from their ABIs:

- Tellor: `getCurrentValue(uint256 _requestId)`
- Chainlink: `latestRoundData()`
- Band Protocol: `getReferenceData(string _base, string _quote)`
- DIA: `getCoinInfo(string name)`

Below is a table of initial results. Note that for the Band Protocol, I had a bit of trouble communicating with their smart contract to get the values for AMPL, so I marked it as `-1`.

`Real-time Price Data as of 7/8/2021, at 11:45 PM:`



	Name	BTC/USD	ETH/USD	AMPL/USD
0	Chainlink	32,796.1669	2,094.6252	0.9180
1	Tellor	32,685.5800	2077	0.9423
2	DIA	32,716.3171	2,082.8340	0.9884
3	Band	32,836.9000	2,095.0900	-1

4.2 Change in Price Feed over Time

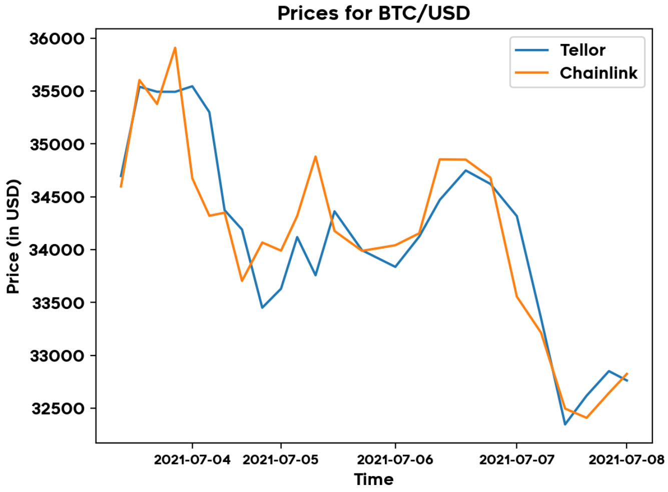
The next metric I decided to look at was the change in value over time of a certain cryptocurrency. Due to the limitations of specific oracles and their smart contracts, I was able to grab historical data from only Tellor and Chainlink.

As a general methodology, I first grabbed the latest request for a value for each price feed. For both Tellor and Chainlink, the respective functions from the respective ABIs returned round IDs. Thus, by subtracting 1 for each previous round, I was able to iterate over the previous 5 days by grabbing the value of a price feed at a specific ID or timestamp.

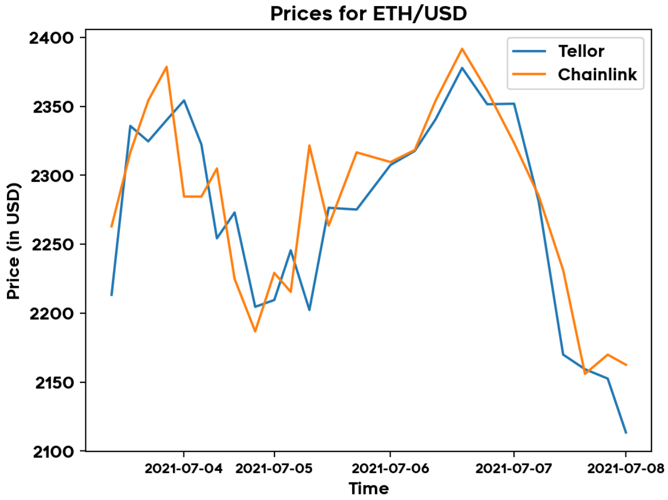
Below are the functions I utilized from each ABI:

- Tellor: `getDataBefore(uint256 _requestId, uint256 _timestamp)`
- Chainlink: `getRoundData(uint80 _roundId)`

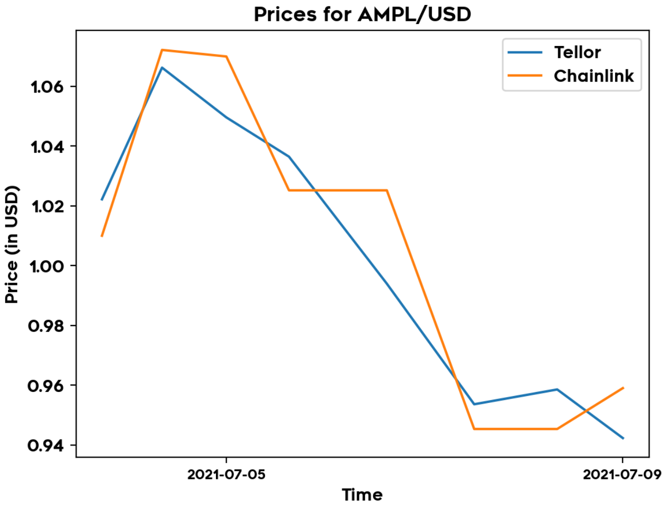
```
Historical Price Feed for BTC/USD:
```



```
Historical Price Feed for ETH/USD:
```



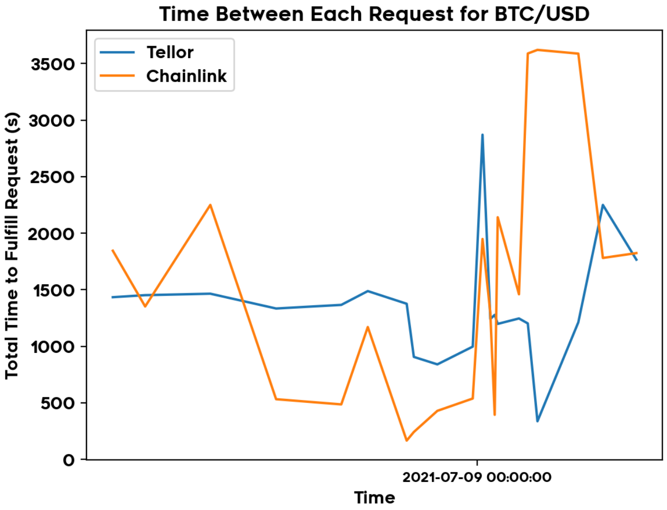
```
Historical Price Feed for AMPL/USD:
```



4.3 Average Time Between Each Request

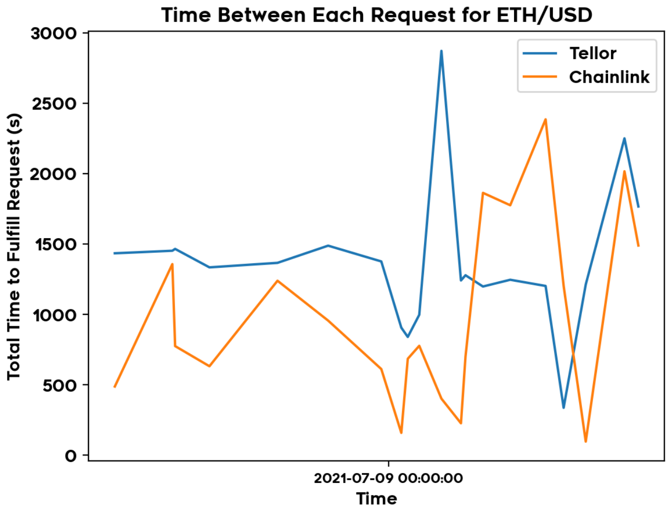
Next, I decided to investigate the time that it took to satisfy each request for the updated value of a price feed. Between requests, I would calculate the time difference between the Unix timestamps of when the value was previously updated to when the next requested was started.

`Average Request Time Graph for BTC/USD:`



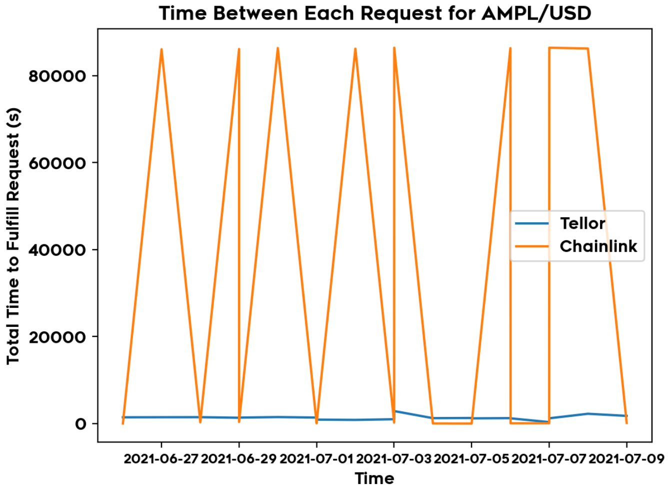
Average time in between each request for Tellor: `1362.95 seconds`
Average time in between each request for Chainlink: `1532.6 seconds`

`Average Request Time Graph for ETH/USD:`



Average time in between each request for Tellor: `1362.95 seconds`
Average time in between each request for Chainlink: `991.9 seconds`

`Average Request Time Graph for AMPL/USD:`

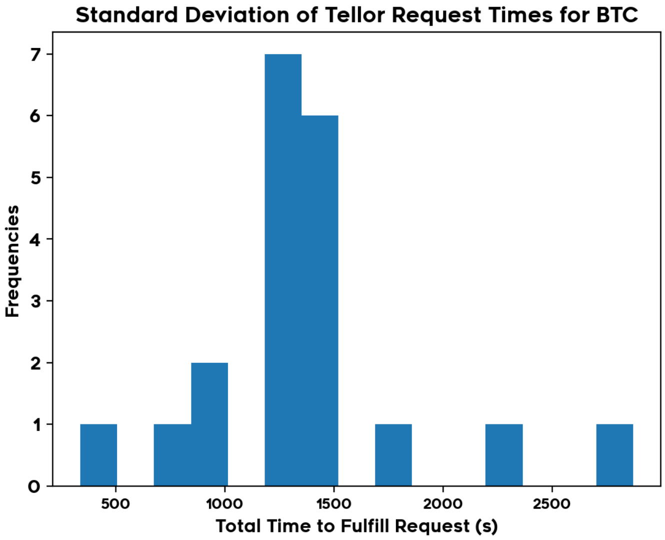


Average time in between each request for Tellor: `1362.95 seconds`
Average time in between each request for Chainlink: `34557.45 seconds`

4.4 Histogram of Time Requests

Note that due the economics of tokens like Tellor, looking at absolute speed is not the most important metric. If anything, a lack of erraticness + a commitment to consistency is much more consequential. Thus, in addition to looking at time per request over time + averages, one could also plot the distribution as a histogram and analyze the standard deviation.

Histogram of Time Requests for Tellor:

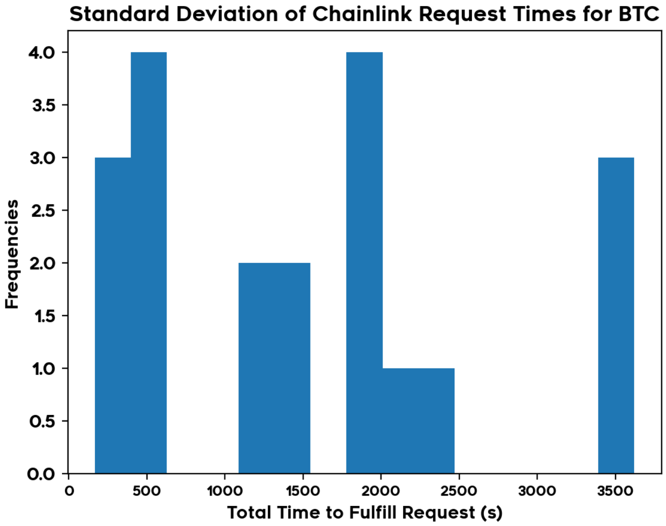


Mean of Tellor times: 1362.95 seconds

Median of Tellor times: 1306.0 seconds

Standard Deviation of Tellor times: 502.1955 seconds

Histogram of Time Requests for Chainlink:



Mean of Chainlink times: 1532.6 seconds

Median of Tellor times: 1406.0 seconds

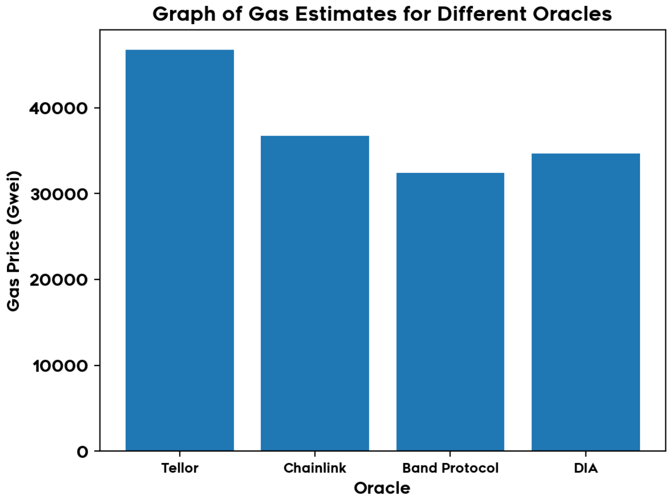
Standard Deviation of Tellor times: 1083.43169 seconds

4.5 Gas Prices

Another metric I decided to investigate involved analyzing the gas estimates for retrieving data from each specific oracle. Simply put, gas refers to the cost needed in order to perform a transaction on a blockchain network. Transactions fees are equal to the product of the units of gas used and the price per unit. However, the units of gas is ultimately fixed [11]. In order to calculate each gas price, I utilized the `estimate_gas()` function within `web3.py`.

It's also important to note that the amount of gas required is also highly dependent on the amount of data being sent back from the smart contract and the function being called.

```
Gas Estimates for Different Oracles:
```



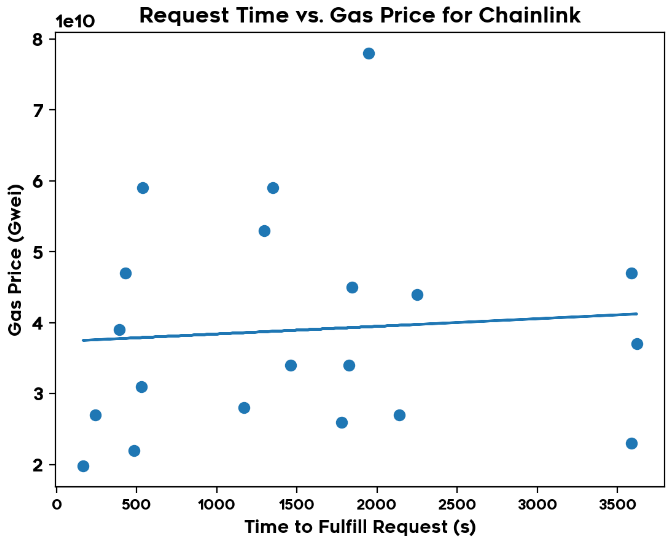
Average Gas Price for Single Value Request for Tellor: `46707 Gwei`
Average Gas Price for Single Value Request for Chainlink: `36707 Gwei`
Average Gas Price for Single Value Request for Band Protocol: `32376 Gwei`
Average Gas Price for Single Value Request for DIA: `34686 Gwei`

4.6 Change in Request Time vs. Gas

When looking at request time, the last relevant portion to note is how the time to fulfill each request changes with fluctuating gas prices. Note that changing gas prices are due to increased demand on the blockchain.

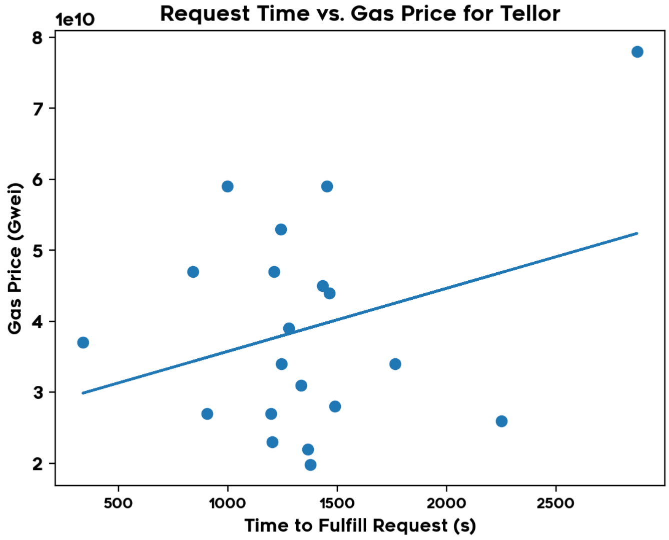
In addition to plotting the scatterplot, the equation for the best-fit line, found using regression, is calculated. From this data, we can also provide the corresponding correlation and p-value. A smaller absolute value of the correlation indicates that there is less of a relationship between gas prices and request times.

```
Scatter Plot of Request Time vs. Gas for Chainlink:
```



Linear Regression: `m = 1077389.578, b = 37338792736.125`
Pearson Correlation Coefficient: `r = 0.07946333`
P-Value: Coefficient: `p = 0.7391223`

```
Scatter Plot of Request Time vs. Gas for Tellor:
```



Linear Regression: `m = 8880913.6157, b = 26885758787.465565`
Pearson Correlation Coefficient: `r = 0.3036148`
P-Value: Coefficient: `p = 0.193136727`

5 Works Cited

[1] https://medium.com/@teexofficial/what-are-oracles-smart-contracts-the-oracle-problem-911f16821b53
[2] https://www.coindesk.com/what-is-an-oracle
[3] https://tellor.io/
[4] https://chain.link/
[5] https://bandprotocol.com/
[6] https://diadata.org/
[7] https://etherscan.io/
[8] https://www.youtube.com/watch?v=F$_{l}4HycnHAI$t=42s
[9] https://web3py.readthedocs.io/en/stable/index.html
[10] https://github.com/cpondoc/oracle-diff
[11] https://www.investopedia.com/terms/g/gas-ethereum.asp