Algorand Analytics Report

Joseph Lavelle
June 6th 2023

Abstract

This report describes the steps taken to create an application to query the Algorand V2 Indexer for block and transaction data and generate relevant visualizations that describe KPIs of the network. It will describe the methodologies and thinking behind the design decisions that were made, as well as how another user should go about setting up the application themselves and using it. Lastly, the report will also describe results and conclusions that were observed based on the application's use regarding the Algorand Network.

Application

Usage

Usage and setup instructions are provided in greater detail in the repository's README.md file. Please consult that document for a more in-depth explanation.

In order to set up the repo, the user must have Poetry and Python 3.11+ installed (earlier versions of Python may also be compatible, however, it was written in 3.11). Once the user has cloned the repo. They will be able to check the pypoetry.toml file to view the dependencies required for the application to run. Running a 'poetry install' command inside the repo will create a Python virtual environment containing all the required dependencies. Then the user must activate their virtual environment with 'poetry env use python3'. Once finished, the app.py file can be run using 'poetry run python app.py'.

The app is a command line interface (CLI) that allows the user to pull new data from the Algorand V2 Indexer using the 'v2/block/ {round number}' endpoint. The app can then use that data to generate plots that help to expose the behaviour of the network. When the app opens, the user may enter "q", "g", "help", or "exit". Any other input will result in the prompt repeating. The "help" command will list the commands available to the user, "q" is used to initiate a new query, "g" is used to generate graphics from the currently stored data, and "exit" is used to exit the application.

When the user starts a new query, they will first be prompted if they wish to overwrite the current data. If they continue, they will be asked for a starting block and an ending block in integer format. Once supplied the app will run a series of queries against the endpoint to retrieve block and transaction data. For larger queries, this can take several minutes due to the 10 requests a second limit on the API. The retrieved data is then written out as a CSV to the data/ folder.

Once data has been saved, the user can now generate graphics using the app. Once back at the app's main menu, entering "g" will run multiple transformations on the dataset and output

multiple plots with relevant information to the data/images directory. It is worth noting that this function also overwrites the currently stored graphics, however, there is no prompt due to the relatively short run time of this function compared to the query.

Repository Structure

The repository is split into several sections. The relevant Poetry files are in the root repository directory, tests are inside the Tests directory, and the core code lives inside the AlgorandAnalyzer directory. Inside that directory lives the main app.py file. Helper functions used by the main app are located inside the modules folder, while all the outputted data/results are contained inside the data folder.

Data and Code Logic

The first dataset that the app must process is the block data retrieved from the Algorand Indexer. The response itself is returned as a single JSON object representing the requested block with all the transactions contained as a nested field. First in order to acquire the data, a while loop is used to query every block in the supplied range against the API while monitoring that the rate limit of 10 requests/second is not exceeded. Due to the nature of the KPIs that the app is to analyze, the decision was made to store the response data as a data frame of transactions rather than a data frame of blocks. This is due to it being a simpler dataset to use for exposing KPIs such as transactions per second due to the lack of nested objects. Transactions of type 'payment-transaction' and 'asset-transfer-transaction' are kept, while all others are discarded. The data is stored as a CSV with the following columns: ["block number", "timestamp", "genesis-hash", "tx-type", "round-time"]. If a block has no transactions, that block is still captured in the dataset in the form of a single row with the 'tx-type' and 'round-time' columns empty. The task originally required the app to store the header for each block, however after reviewing the desired KPIs it was determined that this information would not be necessary to achieve the desired visualizations.

In order to process the data for the visualizations, a series of transformations are required. Firstly, to capture the transactions per second (TPS) metric, the minimum and maximum block timestamps were subtracted together to get the full range of time across the queried blocks, the total number of transactions was then calculated by dropping any rows will empty values and taking the length of the resultant data frame. By dividing the number of transactions by the number of seconds elapsed, the result is the average transactions per second for the whole query.

In order to monitor how the TPS varied over time, a rolling window function was used on the dataset. The data is first grouped by block with a resultant count column describing how many transactions were included in that block. Then a rolling window function was used to determine the amount of time between subsequent blocks being committed (block interval). The transactions column is then divided by the block interval value to obtain the TPS of each block. This data was then graphed against the timestamp to show how the TPS varies with time.

As a separate figure, the block interval was also graphed against time. However, due to the high sample size, the results from this graph could be hard to distinguish because of the large sample size of block intervals with values of 4 and 5 seconds. In order to make this information more readable, a second plot was created, restricting the y bounds of the graph to [1,7.5] and the x bounds to 1/20 the range they were before. This made it much easier to understand what the plot was displaying. The last figure created was graphing block interval against transaction count. This data was already exposed from the last set of transformations making it straightforward to graph.

Areas for Improvement

Due to the time limit on the project alongside other factors such as work and personal life, the design and approach taken to this app was similar to a minimum viable product. While the application fulfills its purpose in querying the requested data and providing insights to KPIs, given more time, there are a multitude of improvements that would significantly improve it.

Currently the error handling and testing of the app is limited. There are a relatively small number of unit tests and there are still some sections of code that could throw unexpected errors.

On top of that the design could likely be more robust and maintainable. More of the steps in the script could be broken into sub functions to better test and verify that all is performing well and ensuring that future changes don't unexpectedly break functionality. In particular the transformations applied to the queried data are not tested inside of the scope of the project as they are not deferred to their own functions. While all of this code was tested ad-hoc during development, functionalizing it and including tests would make it much more reliable and maintainable.

In terms of design, the csv data store is functional and easy to implement, however a proper noSQL database would improve scalability and performance. On top of that, the current inability to persist data automatically across multiple runs is a large, short coming. Historical queries should ideally be saved when they can take several minutes to run due to an API rate limit.

The analysis itself is mostly sufficient. However, it leans heavily on qualitative analysis rather than quantitative analysis. While many correlations/non-correlations (block interval time vs transactions per second etc…) are very visible in the plots, it would've have been better to supply some more numeric data alongside them. It also would have been good to label outlier points on the graphs to better drill down and investigate what caused those outlying points.

Currently, I primarily write python for personal projects and to help me do small tasks. While I have enterprise experience with similar languages (R, Matlab), I was not familiar with the specifics of some of the tools used (Poetry, PyTest etc…) which slowed down my development

and led to some inconsistencies with best practices (such as repo structure). Given more time and familiarity with tools, there is lots I could do to improve the project.

Results

The results in this section were obtained from analyzing a query containing all the blocks within the range 20,923,000 and 20,926,100. This comprises all blocks committed between roughly 2 PM to 6 PM on May 10th 2022 GMT. The methods for acquiring the results listed below are described in the Data and Code Logic section. This will discuss the results themselves and their implications/meaning.

Transactions Per Second

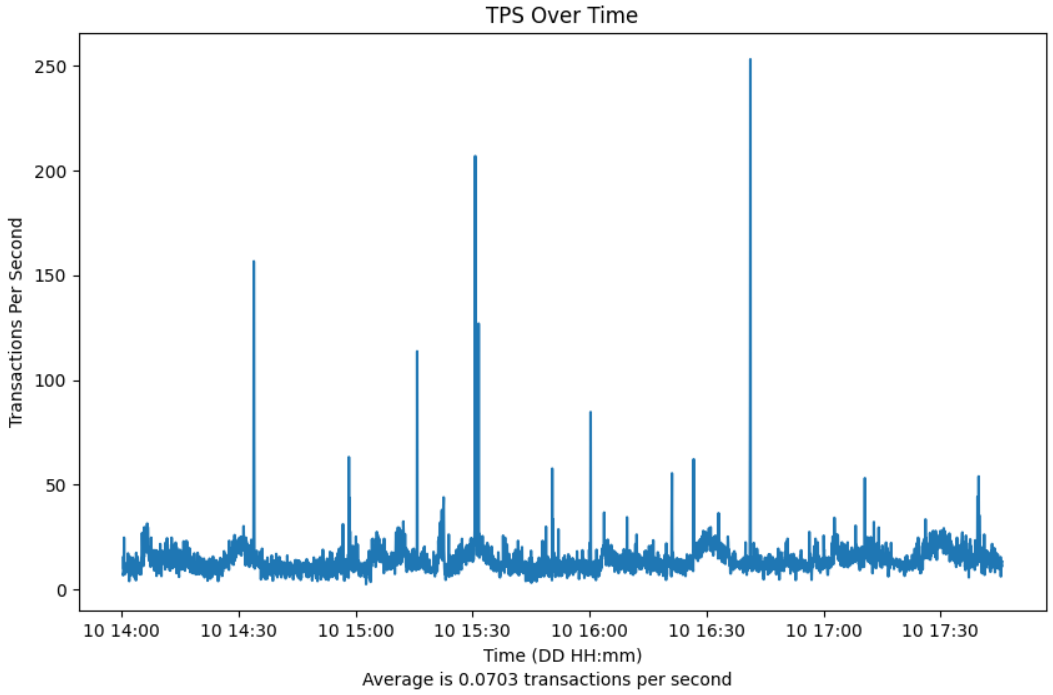

Figure 1 - Transactions Per Second vs Time

Through the methods discussed earlier, the average transactions per second value was calculated to be 0.0703 transactions per second for the provided time range. When the TPS was isolated per block and graphed across the time range, it clearly varied significantly from a high of nearly 250, to a low of near 0. This is intuitive as the speed blocks are processed is fairly constant, while transaction count will vary with people's schedules / algorithmic trading strategies.

Block Intervals

The block interval time is an important metric for measuring the health of the network. The slower blocks are processed, the slower transactions are completed and the less performant the network. To observe how block interval time varied, it was plotted against time in Figure 2.

Initially, it is clear that there are some anomalies in the data. However, it is difficult to determine what is happening with most of the points due to them all clustering at 4 and 5 seconds. A second plot was generated with a smaller x-axis to better understand what is happening there. In the zoomed-in chart shown in Figure 3, it is observed that the majority of the blocks in the query took 4 seconds to be processed, while a significant amount took 5

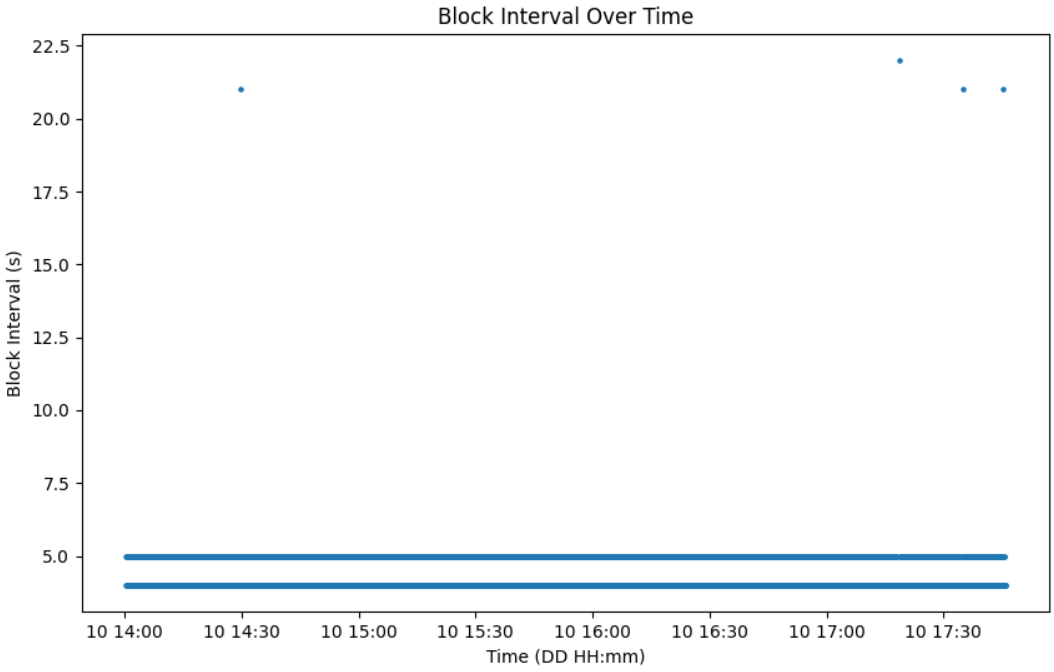

Figure 3 - Block Interval Time vs Time

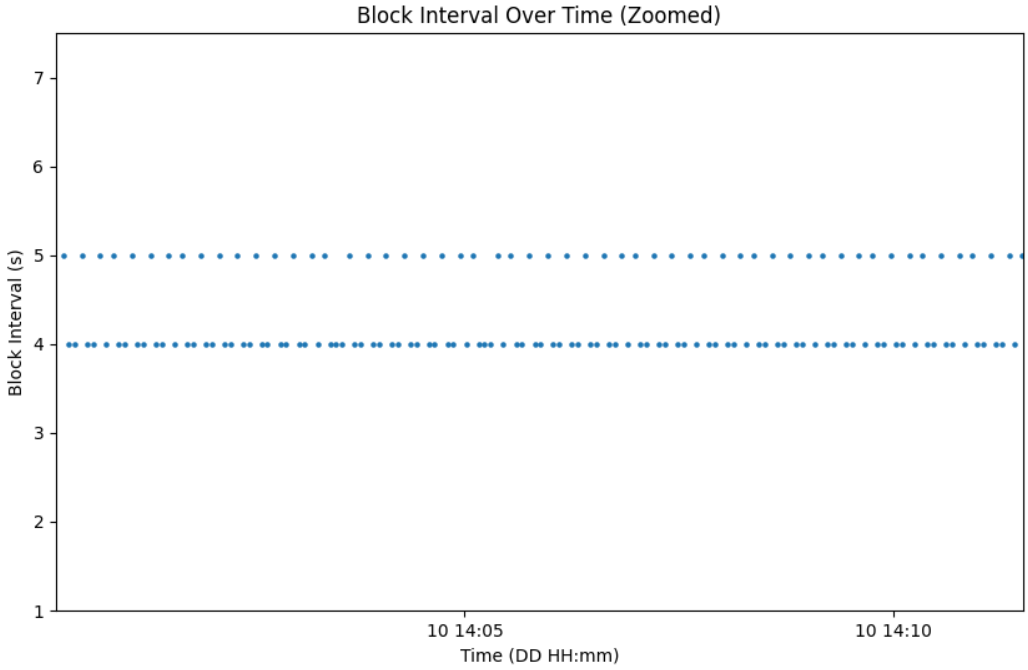

Figure 2 - Block Interval Time vs Time (zoomed)

seconds. This clustering is because the API provides timestamps with a granularity of 1 second. In reality, these values would be much more dispersed within the 3.5 to 5.5-second range.

Lastly, it was investigated if the block interval time was correlated with the number of transactions in the block. These 2 stats were graphed against each other yielding Figure 4. Here it can be seen that block interval time does not seem to be heavily correlated with the number of transactions contained in the block. Even despite transaction counts over 1000 the block interval time remains either 4 or 5 seconds.

What is interesting to note is the presence of outliers in the blocker interval time. In both Figure 2 and Figure 4 it can be seen that there are 4 blocks that took significantly longer to process. The data available does not help to explain why those blocks took so long to process as they have low transaction counts and don't appear irregular otherwise.

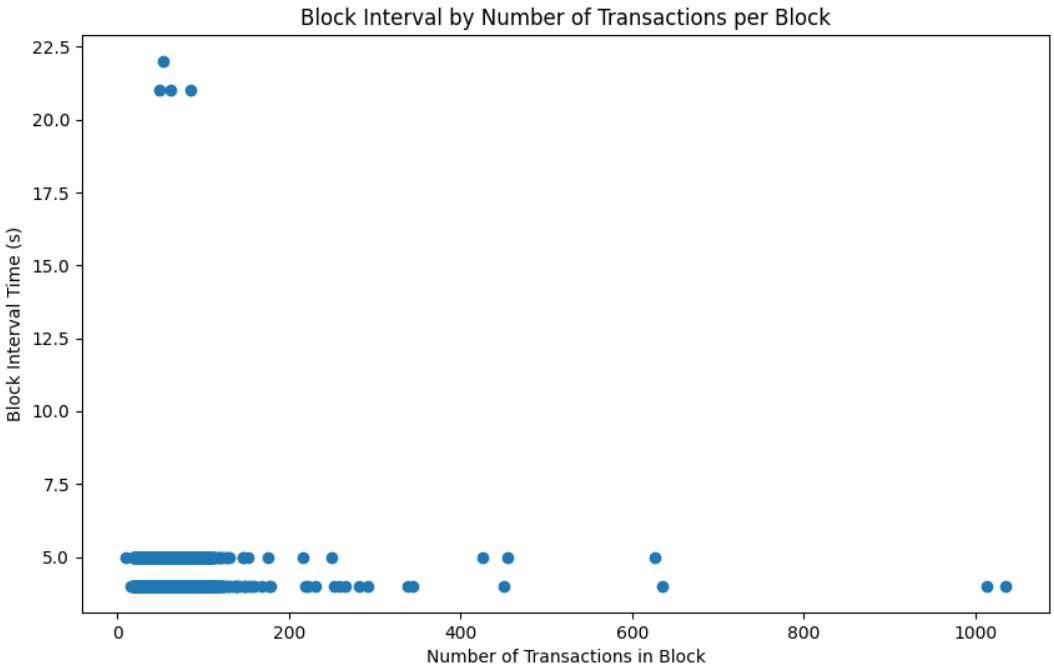

Figure 4 - Block Interval Time vs Number of Transactions in Block

Community Engagement

To engage with different members of a blockchain community and collect feedback for the Metrika platform, it is essential to establish a strong presence within the community and foster open communication channels. Some steps that can be taken to accomplish this include:

1. **Participate in Blockchain Community Events** – Actively participate in conferences, meetups, webinars, and workshops related to blockchain technology. Engage in discussions, share insights, and showcase the value that the Metrika platform can provide to the community. This involvement will help you connect with key community members and establish yourself as a trusted and knowledgeable resource.

2. **Establish a Community Forum** – Set up an online community forum or discussion board dedicated to the Metrika platform. This forum can serve as a central hub for community members to ask questions, share experiences, and provide feedback. Assign a dedicated team to monitor and actively participate in the forum, promptly addressing user inquiries and concerns. Encourage community members to share their thoughts, suggestions, and ideas for improving the platform.
3. **Metrika Client Surveys** – Prompt both current and prospective Metrika users with an open communication path to communicate pain-points and wants of current users and needs that could make prospective users join the platform.
4. **End User Communication –** While the main clients of Metrika will largely be the networks themselves or miners, it is important to also address users. Even if the users themselves may not have a direct need for anything Metrika provides, by understanding issues that upset end users of blockchain networks, Metrika can help identify what improvements clients might need to solve, even they themselves don't know yet.