

Automated Just-In-Time (JIT) Liquidity Provision on the Uniswap Protocol

Oleg Golev (COS)
Jonathan Lee (ECO)
Moin Mir (COS)

"We pledge our Honor that this paper represents our own work in accordance with University regulations"
- *Oleg Golev, Jonathan Lee, Moin Mir*

Abstract

The volatility of the cryptocurrency market presents an issue for investors who want to execute large transactions. Just-in-time (JIT) liquidity is an approach that attempts to address this issue. JIT liquidity providers monitor the Ethereum mempool for sizable transactions, inject assets into the associated liquidity pool to fill that transaction's liquidity needs right before it executes, and then withdraw the assets from the pool alongside the accrued fees, with all three transactions being done within the same block (using Flashbots). This approach improves the efficiency of a transaction, reduces the risk of price slippage, and can make some money for the JIT liquidity provider. In this paper, we describe a system our team built to (1) watch the Ethereum mempool for Uniswap V2 transactions in real time, (2) infer which transactions are viable for a JIT liquidity attack, and (3) monitor the statistics, including the potential profit, of executing a JIT liquidity attack on the identified transactions using a Python dashboard. Our system can help potential traders to make better decisions about the use of JIT liquidity and improve their overall trading strategies.

Keywords: Just-In-Time Liquidity, Uniswap, Simulation Model, Transaction Efficiency

All code is publicly available on our [GitHub](#) alongside a [README](#) on how to run everything.

1. Introduction

1.1. Background on JIT

The crypto market is well known for its volatility due to its decentralized nature. Just-in-time (JIT) liquidity is a strategy that helps address this problem. The JIT strategy is where a liquidity provider (LP) adds a lot of value to a token pair's liquidity pool right before a large swap for that pair. The LP then removes the liquidity from the pool alongside the accrued fees after the large swap executes.

This way, JIT is a viable money-making strategy for LP providers and helps traders and investors by (1) providing immediate access to liquidity as needed, (2) mitigating the effects of impermanent loss, and (3) improving efficiency of larger swap transactions.

While JIT provided ~2 billion dollars worth of liquidity to traders, Uniswap V3 supported trading volumes of over 600 billion in the same period, indicating that JIT filled only around 0.3 percent of the total liquidity demand [4]. This implies that JIT is, while viable, still a niche strategy reserved for only a small subset of all transactions.

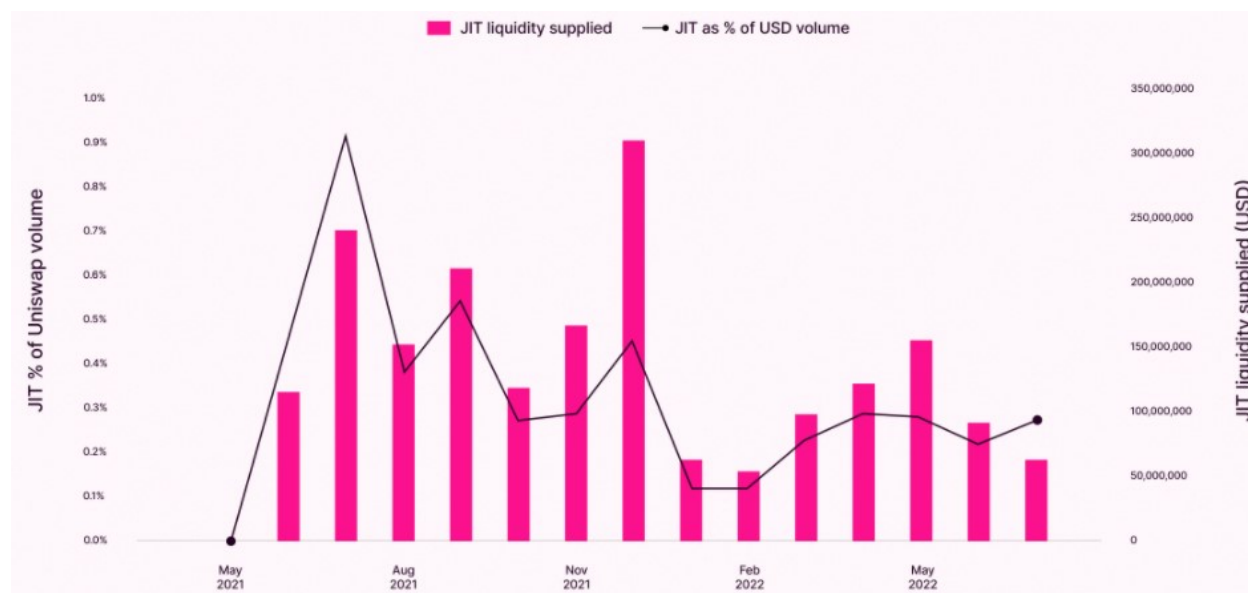


Figure 1: Percent of liquidity provision via JIT to Uniswap Volume (May 2021 to July 2022) [4].

1.2. How JIT Works

JIT liquidity is a specific type of liquidity provisioning (LP) strategy where a liquidity provider provides funding to a borrower right when it is requested. There are four components for executing JIT liquidity provision to a transaction:

1. First, a liquidity provider observes a pending swap transaction in the public mempool (transactions of interest are usually valued in the millions of USD).
2. The liquidity provider adds a large amount of assets to the pool of the token swap pair.

3. The transaction itself then executes using the above-provided assets. The provider would receive a fee for providing this liquidity to the swap.
4. After the swap occurs, the liquidity provider removes all liquidity they added to the pool.

JIT liquidity provision is enabled by Flashbots which allows multiple transactions to be bundled and executed together in one block. In the case of JIT, the bundle would contain (1) the add liquidity transaction, (2) the target swap transaction, and (3) the remove liquidity transaction.

Action	Details
Add liquidity	0x769f... adds >11M USDC to USDC/ETH 5bps pool
Swap	0x1261... swaps ~1500ETH for ~7M USDC in this pool. Note that the LP's position changed from 11M USDC to 5.5M USDC + 1200 ETH
Remove liquidity	0x769f... removes ~5.5M USDC and ~1200 ETH

Figure 2: Example Flashbots bundle created to execute a JIT liquidity attack [4].

1.3. Why Use JIT

One of the most significant risks for liquidity providers on Uniswap is the potential for impermanent loss. This happens when an asset's price in the pool alters such that it is not reflected in the external market. More specifically, when one of the asset's price in the pool changes relative to the other asset, the pool's ratio alters as well. This can be the consequences of market fluctuations, unexpected events, etc. The impermanent loss is then the difference between what the LP receives and what they would have received if they had not invested their liquidity [2]

Let \mathbf{P}_0 be initial Price and \mathbf{V}_0 be the value of 2-asset (\mathbf{x}, \mathbf{y}) portfolio. \mathbf{V}_0 is then given by:

$$\mathbf{V}_0 = \mathbf{x} + \mathbf{P}_0 \cdot \mathbf{y} \quad (1)$$

Then \mathbf{V}_1 would be the the new value of the portfolio when being traded at new price \mathbf{P}_1 :

$$\mathbf{V}_1 = \mathbf{x}' + \mathbf{P}_1 \cdot \mathbf{y}' \quad (2)$$

The impermanent loss ε can then be defined as:

$$\varepsilon = \mathbf{x}' + \mathbf{P}_1 \cdot \mathbf{y}' - (\mathbf{x} + \mathbf{P}_0 \cdot \mathbf{y}) - (\mathbf{x} + \mathbf{P}_1 \cdot \mathbf{y} - (\mathbf{x} + \mathbf{P}_0 \cdot \mathbf{y})) \quad (3)$$

$$= \sqrt{\mathbf{P}_1 / \mathbf{P}_0} - \frac{1}{2} \cdot (\mathbf{P}_1 / \mathbf{P}_0) + 1 \quad (4)$$

Full derivation of the impermanent loss can be found in Appendix I of [2].

JIT Liquidity reduces the risk of impermanent loss by only providing liquidity when there is

a demand for it. As a result, liquidity providers can avoid situations where the underlying assets' prices change significantly when a corresponding change in the external market does not occur.

1.4. Traditional vs. JIT Liquidity Provision

Alice is a liquidity provider where her current position in pool **A** is P_0 . Bob utilizes pool **A** as a result of swapping the pool's tokens. Consequently, the price changes from P_0 to P_1 , where $P_1 > P_0$. Alice will thus make profits from the swap fees based on $P_1 - P_0$.

Now suppose that Alice is still a liquidity provider with some stake in pool **A**, and Bob ends up performing a swap transaction that utilizes pool **A**. Third actor Justin notices that Bob has submitted a transaction to swap tokens within a transaction mempool. In response, Justin submits a transaction that would inject a lot of liquidity into the pool **A**. To ensure that his liquidity gets added before Bob's transaction gets processed, Justin increases his gas prices. In other words, Justin offers a significant amount of liquidity at the current price P_0 , while Alice has provided a comparatively lower amount. Thus, when Bob's swap is executed within this price range, Justin earns the majority of the fees, and the price moves up from P_0 to P' , where $P' - P_0 < P_1 - P_0$ due to the greater available liquidity (due to Justin's injection) at that price point.

1.5. Hedging

Hedging is a strategical risk management method that minimizes potential losses. In the context of our project, hedging can be used in a JIT liquidity attack to reduce the risk of loss in the case that an attack is not successful or the market moves against a provider's position [4].

There are a number of ways that a JIT liquidity provider can approach hedging:

1. In short-selling, an investor would borrow cryptocurrency from another source and sell it in the market, with the expectation that the price falls sometime in the near future. When the price falls, the provider can buy back the cryptocurrency at a lower price and return it to their lender. In a JIT liquidity attack, the provider could sell some of their holdings in case the price drops.
2. A put option gives an investor the right to sell a cryptocurrency at a specified price by a specified date. A liquidity provider could protect themselves against a price drop by purchasing options on the cryptocurrencies for which they plan to provide liquidity [1].
3. A futures contract is an agreement to buy or sell an asset at a predetermined price and date in the future. By entering a futures contract, the liquidity provider could protect themselves against a price drop [3].

For this project, our team did not model a hedging strategy since this was beyond the scope of our simulation. The goal of this project is to build a high-concurrency real-time transaction feed and evaluate transactions for JIT liquidity attack viability. We decided to include a section about hedging since it is often a core component of many investment strategies, including a JIT LP strategy, if deployed by a trading firm in a real-world scenario.

2. Design

2.1. System Overview

This project concerned itself with building and assessing the system displayed in **Figure 3**. We used Rust to write a robust and low-latency service that listens for new pending mempool transactions from an external Ethereum node (in this case one of Infura’s nodes) in real-time, filters for Uniswap V2 swaps, and then makes all identified swaps available via a REST API to other services. In our case, we built a real-time data analytics dashboard in Python to consume and present different pieces of information related to the swaps identified by the Rust infrastructure.

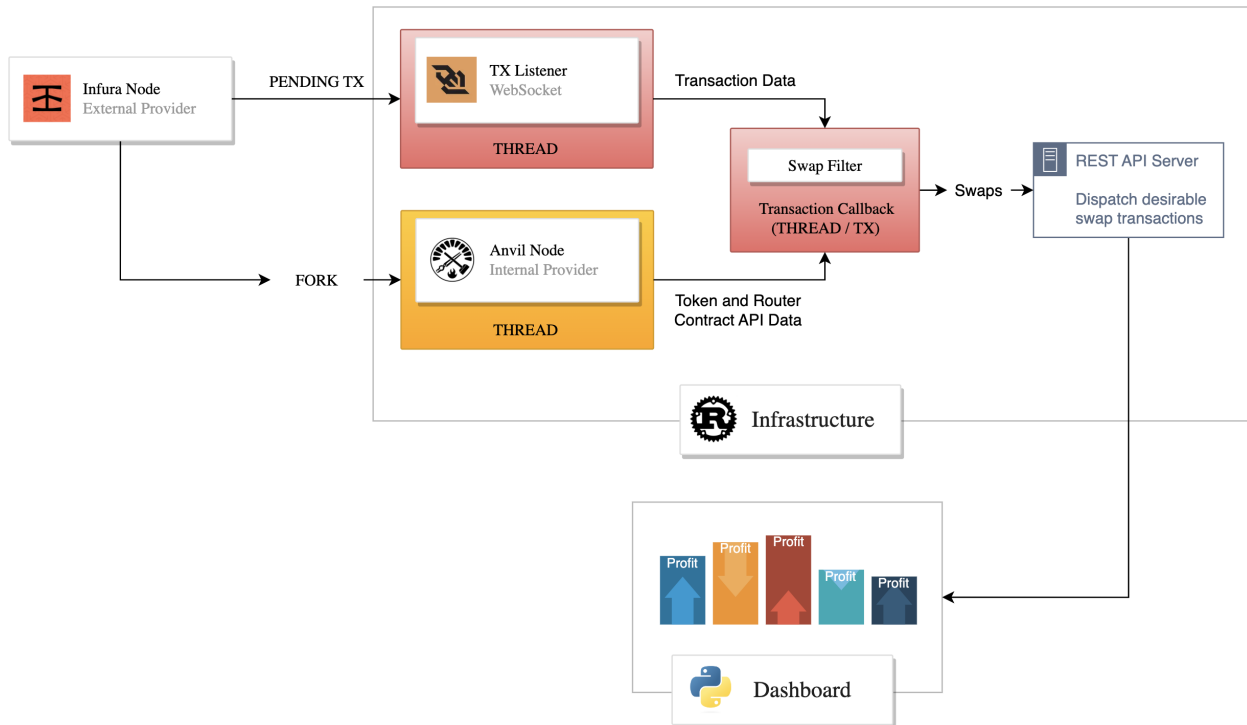


Figure 3: Complete diagram of the designed and engineering system.

2.2. Data Collection

The Rust infrastructure collects multiple pieces of information for each swap to be accessible via the REST API. To reduce reliance on the external provider, we start a local Anvil node (similar to Ganache), onto which we fork the Infura node contents—including all blocks, transactions, token pairs information, and contracts—upon start-up. The entries marked with an asterisk were acquired from the local Anvil node rather than the external Infura node:

1. The transaction hash
2. The quantity, *reserves, and *symbol of the *from* token
3. The quantity, *reserves, and *symbol of the *to* token
4. The current gas fee estimate
5. Timestamp of when this transaction was received by the system from the Infura node
6. Timestamp of when this transaction was packaged with the data (and therefore would be ready to send as part of a Flashbots bundle if this were made into a real trading strategy).

2.3. Python Dashboard

The Python dashboard was created using Streamlit and uses the Pandas library to provide the user with the following pieces of information:

1. Simple metrics such as the total swaps found, total "valid" swaps, total "viable" swaps, and the mean swap size (detailed descriptions of these metrics will be provided later)
2. Graph of the swap size (in WETH) of all valid transactions seen up to now
3. Graph of cumulative potential profit if JIT was performed on all transaction where it was predicted to be profitable
4. Graph of potential profit of running a JIT liquidity attack per each viable transaction seen in the real-time data feed
5. Graph of system latency over time alongside some useful metrics where latency is computed as the difference between when the transaction was processed (and would be ready for Flashbots bundling) and when the transaction was received by the system from the Infura node.

The dashboard currently shows live data by querying the Rust REST API at the `/get_data` endpoint.

2.4. Estimating Profit

In order to support the functionalities of the Dashboard, we perform a few calculations. The JIT liquidity attacker's revenue is composed of Uniswap fees and the price impact paid by the trader:

$$\text{Gross Profit} = R_{\text{fee}} + R_{\text{pi}} - \text{Gas fee}$$

In Uniswap V2, the available liquidity is determined by the size of the liquidity pool, where liquidity is constant across all price ranges. In Uniswap V3 however, liquidity is not a fixed amount across all price ranges. Instead, liquidity providers can specify certain price ranges (ticks) in which they want to provide liquidity. As such, the revenue is distributed to all LPs based on how much liquidity they provided inside a tick. In cases where a trade is executed in more than one tick interval, the LP receives revenue only for the amount that was traded in the tick interval in which they provided liquidity. As such, the maximum amount of liquidity for which a JIT liquidity provider can accrue fees would be the exact volume of the swap that is being targeted for a JIT liquidity attack. Below is the generic expanded equation describing the profit potential of a JIT attack on Uniswap V3:

$$\text{Profit} = (\Lambda \cdot \phi + \iota) - \Gamma(2\gamma_{\text{approve}} + \gamma_{\text{add_liq}} + \gamma_{\text{remove_liq}}) - \rho \quad (5)$$

where:

Λ = total liquidity supplied (set at size of the swap transaction)

ϕ = Uniswap liquidity provider fee rate

ι = the price impact on the Uniswap pool due to the swap and liquidity supplied

Γ = current gas price

γ_t = the gas cost of a transaction of type t

ρ = miner share

In our system, since we are monitoring the Uniswap V2 pool, we use the following simplified calculation for expected profit:

$$\text{Profit} = \Lambda \cdot \phi - \Gamma(2\gamma_{\text{approve}} + \gamma_{\text{add_liq}} + \gamma_{\text{remove_liq}}) \quad (6)$$

Notably, we disregard the price impact of the transaction as well as the miner share. So the profit we get is a lower bound before the miner share is distributed which has historically been $\approx 60\%$.

3. Results

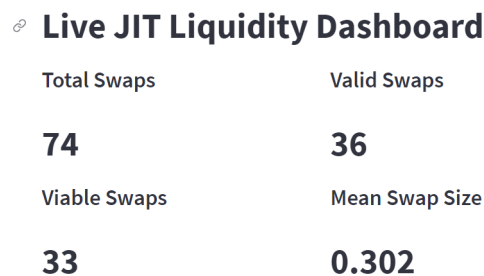
We analyzed the Just-In-Time (JIT) liquidity attack viability on the Uniswap V2 mempool transactions. Specifically, we focused on swap transactions that traded for WETH. Our system, as described above, conducted an analysis of live data to display several metrics and graphs in real-time.

Given the high costs of liquidity provision and removal, JIT liquidity attacks typically target very large swap transactions. For the purpose of illustrating our dashboard, we reduced the simulation's costs to observe more viable swaps and chart activity. However, when actual transaction costs were incorporated into the simulation runs, viable transactions were not observed before our Infura API limits were reached.

The ensuing sections detail our findings derived from one-hour system runs conducted on 5/14/2023. We performed multiple runs across different days and times, which yielded similar outcomes.

3.1. Analysis of Swap Transactions

Our analysis revealed that a significant proportion of swap transactions did not adhere to the Uniswap's constant product formula ($x \cdot y = k$), rendering them invalid. These transactions were thus filtered out from our set of valid transactions. For each of the remaining transactions, we calculated the potential maximum revenue if we were to perform a JIT liquidity attack using equation (6) from above. In order to estimate the associated cost of each transaction, we used publicly available gas price data. We then identified the "Viable Swaps" as those transactions that would yield a profit if subjected to a JIT liquidity attack. Most swaps were small (< 1 WETH), with the observed maximum volume being ≈ 5 WETH across all runs.



Live JIT Liquidity Dashboard	
Total Swaps	Valid Swaps
74	36
Viable Swaps	Mean Swap Size
33	0.302

Figure 4: Swap metrics as displayed on the Python dashboard (costs reduced).

When we used the actual transaction costs for adding and removing liquidity, we did not find **any** viable transactions. This reinforces that JIT liquidity attacks are reserved for very large transactions and, as such, for entities with enough capital to cover the liquidity demand for those large transactions to actually accrue enough fees for the attack to be worthwhile.

Live JIT Liquidity Dashboard

Total Swaps	Valid Swaps
150	41
Viable Swaps	Mean Swap Size
0	0.277

Figure 5: Swap metrics as displayed on the Python dashboard (real costs).

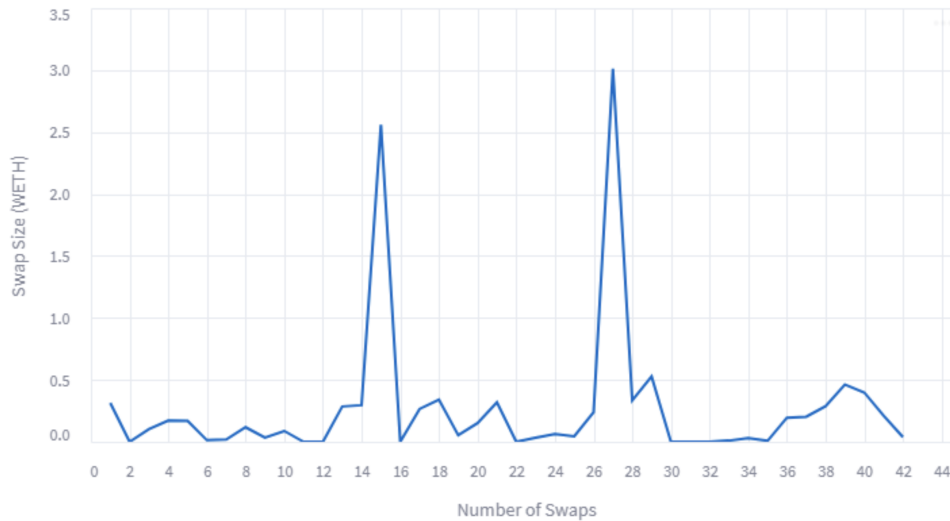


Figure 6: Real-time volume of valid swap transactions from start of Rust service.

3.2. Profit and Loss

The Python dashboard displays both cumulative and per-transaction potential profit if the viable transactions were to be targeted for a JIT liquidity attack. The following graphs illustrate the high cost of JIT liquidity in the real-world, necessitating that transactions be large to be JIT-profitable:

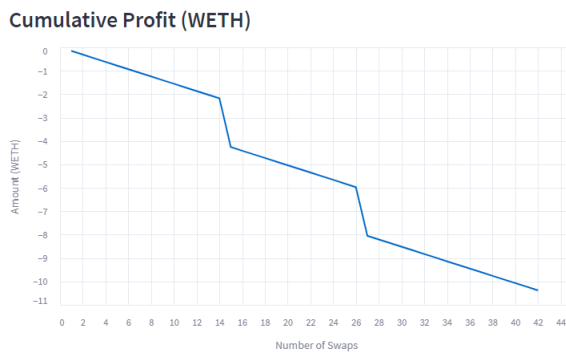


Figure 7: Cumulative profit if every viable transaction was targeted for JIT liquidity.

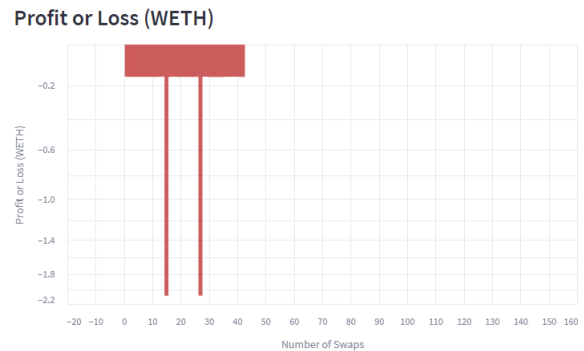


Figure 8: Potential profit if every viable transaction was targeted for JIT liquidity.

Reducing costs yields graphs that would be more indicative of a profitable system. The following are graphs that one could see if they ran the system for long enough to see transactions that would be truly profitable if targeted for JIT liquidity attacks.

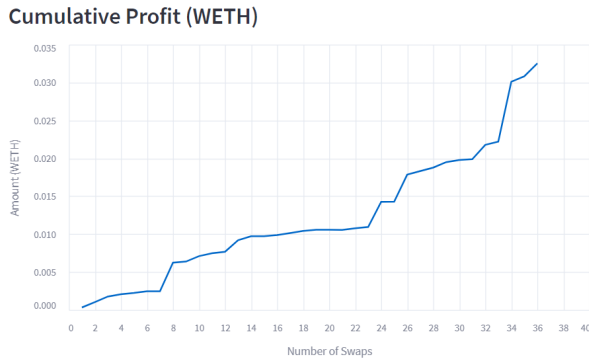


Figure 9: Cumulative profit if every viable transaction was targeted for JIT liquidity (costs reduced)

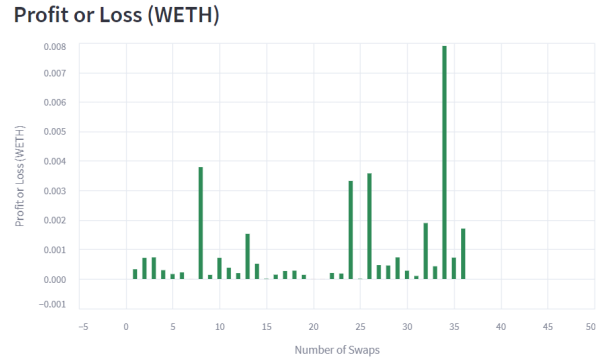


Figure 10: Potential profit if every viable transaction was targeted for JIT liquidity (costs reduced).

3.3. Latency

Since our system does not actually create or send Flashbot bundles, our latency analysis was focused solely on transaction processing. The latency statistics compute the time difference (in milliseconds) between the transaction receipt and the completion of its processing (i.e. when it is available for distribution via the system’s REST API). What is not shown is the interval of time between the receipt of the transaction hash and the receipt of the actual transaction by our system. This interval of time is dependent on the system configuration since transaction data is requested on a polling basis and requires the user to specify the number of times a transaction is requested before giving up and the time interval between these attempts. There is significant room for fine tuning in this area.

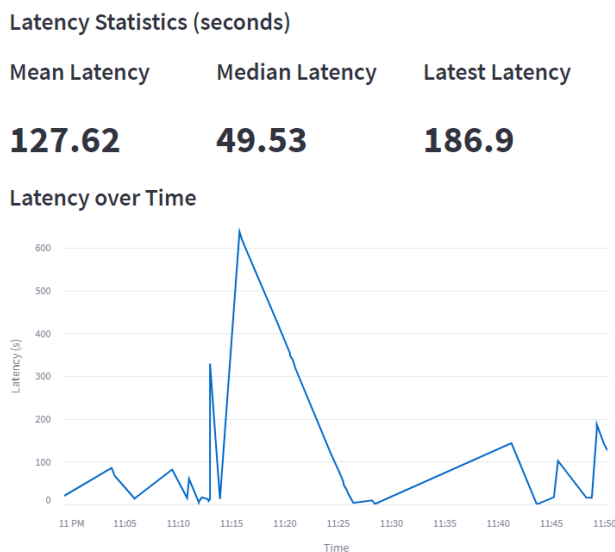


Figure 11: Viable transaction processing latency (in ms) for one-hour runtime of the system.

4. Conclusion and Future Steps

In this project, we thoroughly explored our interest in JIT liquidity provision. Namely, our achievements were tri-fold:

1. We explored further and applied our knowledge of Just-In-Time liquidity provision and its potential as a money-making trading strategy.
2. We built and measured the latency of a highly-concurrent Rust service used to monitor and process Uniswap transactions in real-time
3. We used our knowledge of JIT liquidity theory and gas fees to estimate volume and profitability (in WETH) of valid Uniswap transactions

For this particular project, we focused on Uniswap V2 transactions due to its liquidity pool simplicity. Unlike Uniswap V2, Uniswap V3 supports pools of different fee tiers and token ratios, which would have made implementation much more complicated. Likewise, there is substantial enough difference in API usage required to acquire the desired information for each transaction when dealing with the two protocols. Therefore, transitioning our project to Uniswap V3 would be the first goal for potential future progress. Finally, this project operates as a survey tool but can easily be re-written as a viable trading strategy, given the deploying entity has enough capital to support large JIT liquidity provision for large-volume swaps.

References

- [1] S. S. Adam Hayes, “Short selling vs. put options: What’s the difference?” 2022. [Online]. Available: <https://www.investopedia.com/articles/trading/092613/difference-between-short-selling-and-put-options.asp>
- [2] G. D. Andreas A Aigner, “Uniswap: Impermanent loss and risk profile of a liquidity provider,” 2021. [Online]. Available: <https://arxiv.org/ftp/arxiv/papers/2106/2106.14404.pdf>
- [3] P. G. Helder Sebastião, “Bitcoin futures: An effective tool for hedging cryptocurrencies,” 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S1544612319301849>
- [4] A. A. Xin Wan, “Just-in-time liquidity on the uniswap protocol,” 2022. [Online]. Available: <https://blog.uniswap.org/jit-liquidity>