The Fuel for Fast Execution

An updated thesis on Fuel for the modular world

RainandCoffee
May 19

Preface

Fuel is the most interesting execution environment we have come across and we (Maven11) are proud to be supporters of Fuel. Fuel situates itself as the fastest execution layer for the modular future, which is one that most blockchains now realise they have to move towards, to provide the scaling that is needed for global use. Furthermore, it takes full advantage of the flexibility modularity offers to developers by not replicating the EVM. In this article, we will cover the uniqueness of Fuel and how they're trying to provide the best possible experience for users, developers and block producers. If you're interested in building on Fuel, don't hesitate to reach out to us, we'd love to hear from you

Introduction to Execution Layers

In our last piece, we discussed at length the unique functionalities that Celestia enables. We also touched upon various modular protocols and their architecture. So for this next piece, we're going to be looking at a different type of modular protocol. This is not a modular data availability layer that enables the building of clusters, but rather a modular execution layer.

So what is an execution layer even? An execution layer is a chain that purely handles the execution of transactions, while delegating the rest of the functionalities of a blockchain to other chains, such as consensus/data availability and settlement. An example of an execution layer is the current implementation of rollups on Ethereum, such as Arbitrum and ZKSync. So how is this different from a settlement layer? A settlement layer is also an execution environment, but one that has trust-minimised bridge contracts on it, that rollups use for unified liquidity. These rollups also send their various proofs and block headers to the settlement layer, which acts as the source of truth for the rollups.

If you're unsure of what a rollup is, then let's break it down quickly. A rollup is a scaling solution that operates off-chain from the main L1 (in most cases, Ethereum). This solution performs transactions off-chain, which means it doesn't have to compete for precious block space. After executing the transactions, it will send a batch of transaction data or proof of execution to the L1, where it will be settled. Because of this, the layer 2 scaling solution is secured by the same layer 1 security measures, since the DA layer or settlement layer acts as the source of truth for the rollup.

Optimistic and ZK Rollups on Ethereum

Rollups come in two different flavours primarily (However there are other types of rollups too). These two are ZK rollups and Optimistic Rollups. **Optimistic rollups** assume that transactions are valid by default (hence the name Optimistic). However, in the case of a malicious or wrong transaction, fraud proofs are generated and sent

to the L1 to roll it back, while the transaction proposer gets slashed. **Zero-knowledge (ZK) rollups** run complex computation off-chain as well, through a circuit which provides a validity proof (snarks, starks, plonks, kimchi etc.). This validity proof is posted to the L1 to show that the rollup has executed the transactions correctly, without actually posting the transaction data itself.

Modular Execution Layer

Now that we have established what an execution layer is, and how various current rollup scaling solutions work, let's take a look at what it means to be a modular execution layer. Fuel defines a modular execution layer as ***a verifiable computation system designed for the modular blockchain stack.*** *So what does that actually mean?* It means that Fuel is a blockchain execution environment that is able to leverage a lazy/modular blockchain for data availability. Furthermore, the verifiable computation part of the sentence refers to the flexibility of being either fraud or validity provable.

Note that the DA attestation refers to the proof of DA being sent to a contract on the settlement layer - With Celestiums through Gravity Bridge, or IBC with ICS enabled chains.

So why is Fuel, like many others, moving away from monolithic design principles?

During heavy congestion on monolithic blockchains, L2s or rollup transaction costs rise to outlandish amounts. One of the examples of this happening recently is during the Otherside land sale by Yuga Labs, which catapulted even the transaction costs of L2s to two digits as a result of having to settle transactions on a monolithic chain. This is just one of the reasons why Fuel wants to move away from monolithic designs where consensus, data availability and execution are combined. The execution that Fuel wants to provide will be enabled by data availability and consensus from Ethereum’s rollup centric roadmap, and from other modular protocols such as Celestia. As a result of not being hindered by monolithic entities, Fuel is able to specialize at the execution layer which can immensely increase the capacity of execution. Where Fuel stands out is that they are working to provide fast throughput and complex smart contracts through their own distinctive VM, transaction design and domain-specific language. They do this by optimizing for the future of blockchains, where data availability capacity is no longer the bottleneck.

Fuel

As mentioned earlier Fuel has been a layer-2 protocol for Ethereum, but is also situating itself to become the go-to execution layer for the future of modularity. Fuel's strategy to become the place to-go-to is to provide high throughput of Ethereum-style composable smart contract transactions.

They’re still an optimistic rollup. However, Fuel has a very unique design compared to existing optimistic rollups. The uniqueness comes through their UTXO-based design, which allows for parallel transaction execution, which we’ll get more in-depth with later on. Fuel will also implement the ability for operating nodes to accept fees in other tokens than ETH, if they're willing to accept it for transaction fees. This is done by allowing nodes to set up non-primary mempools, where fees are paid in the token that the nodes want to receive.

Different mempools operated by Fuel Nodes which can choose to accept various tokens for the fee the end-user pays

So let’s dig deep down and analyse how Fuel works.

Optimistic Rollups

The way rollups work is that anyone is allowed to construct blocks off-chain and then submit them to Ethereum as calldata (which we severely lack, hence the development of blob transactions and eventually the move to sharding). This is done through a rollup contract that keeps track of the block headers of the rollup on Ethereum. In the case of optimistic rollups, a fraud-proof can be submitted and roll back the chain, burn the bonded amount and reward the fraud prover.

Now let’s look at how a rollup node operates, both off-chain and symbiotically with Ethereum.

Fuel’s Operating Nodes

Rollup users send transactions to a Fuel node, which is done automatically through whatever dApps you’re using on the rollup side. The client/sequencer will then roll up transactions together into a Fuel block, which then gets sent to Ethereum, which confirms the block. The client also handles deposits from Ethereum, which is what I’ve called a bridge contract previously, which enables unified liquidity between the rollup and Ethereum.

So how does Fuel’s fraud proofs work which ensures no malicious actions?

Once a fraud proof is submitted to the rollup contract, it needs to be validated to check that it has the correct formatting. This is done so that it can ensure that the proof wasn’t maliciously sent. Since if that was the case, a valid block could then be reverted as the result of the illegitimate fraud-proof. The proof verifier contract verifies that the fraud proof is correctly formatted and valid, and if so, processes it to slash the offending rollup block.

This scheme is unique since it does not require state serialization. Which is the action of computing the Merkle root of the state (which we covered in our previous article) after each transaction or block. This is made possible since Fuel uses UTXOs which means that transactions don’t need to be executed in order. This is because you can simply check at the end that every input that was computed, was previously unspent and unique. As a result of this, transactions are able to happen all at once and do not need to be ordered.

UTXO-based transaction and account design

UTXO stands for unspent transaction output and is the transaction data model used for Bitcoin. UTXO represents an N amount of a coin/token which is allowed to be spent by an account by another. UTXO makes use of public keys to identify and transfer ownership of holdings. UTXO addresses are formatted by a public key which has an associated private key, which allows for spending on that particular account. This allows for having atomic amounts of tokens or state, which can be controlled by a spender.

Bitcoin’s UTXO Model

In a simplified model, each UTXO has two fields: 1. amount of coins, and 2. script hash that defines an owner. A contract UTXO has four fields, 1. amount of coins, 2. contract ID, 3. contract code hash and 4. a storage root—which is quite similar to a normal account-based contract. Keep in mind that, unlike non-contract UTXOs, contracts don’t have a defined owner similarly to contracts on Ethereum. Since each

contract UTXO is uniquely identified by its contract ID in addition to its UTXO ID, each contract can be used multiple times in a single block by simply referencing it by contract ID.

The reason for using UTXOs is that it enables something incredibly powerful, which is parallel transaction execution. This is because transactions have no interdependency and as such can be executed in parallel with one another. While this was tried previously, it depended on users signing over on the effects of the transaction being spent, resulting in contention. However, by not forcing users to sign every effect of a transaction, you can allow for much more streamlined execution. Since the atomic nature of UTXO transactions define each area of state a transaction is going to, it allows nodes on Fuel to determine which transactions have no interdependency and thus execute them in parallel.

This obviously makes Fuel extremely unique, since no other optimistic layer-2 uses a UTXO-based transaction system. But this is also their strength since it allows for parallel transaction validation, which should increase scalability compared to account-based rollups.

So why is it that using a UTXO based transaction system provides such incredible scalability? This is because each transaction can spend and process up to several inputs and outputs.

Several Inputs and Outputs are enabled in each transaction, as a result of using a UTXO-based transaction model

As a result of this, it allows Fuel to conduct atomic multi-user transactions, which unlocks a new world of possibilities for dApps on top of Fuel. On Fuel, each input in a UTXO-based system is only produced and consumed once, which means that the rollups chain's state is stored as a key-value.

Since computing state elements with UTXO IDs is deterministic and stateless, once consumed, it allows for long chains and 'trees' of pre-signed transactions as well.

A 'Tree' of pre-signed transactions that is able to go to various IDs

As a result of the state database only needing to be checked for the existence of consumed elements, it allows for interesting possibilities and interactions between users and transactions.

As with Bitcoin instead of having accounts you have atomic amounts of x coin/token/state that are controlled by a spender. These state elements can represent different assets, such as allowing ETH and ERC-20 tokens. Furthermore, because of Fuel's various spending conditions, you can create HTLC outputs (allows for one cryptocurrency to be traded for some amount of cryptocurrency on another blockchain) which will allow for instant withdrawal between Fuel and Ethereum for example. This is done through LPs that can offer up liquidity to users wanting to withdraw quickly, for a fee of course. However, there's zero systematic risk if they fully validate Fuel blocks before executing the withdrawals.

So to sum up why a UTXO-based design is distinctive:

1. Parallel transaction validations
2. Unique fraud proofs without requiring state serialization

Security

The security of a blockchain can be defined as the cost of attacking the network's history. In most cases making it prohibitively expensive to attack the network is what secures it—cryptoeconomic security.

In the case of a Layer 2, such as Fuel, the rollup must also be trustless with state liveness and state safety. This is the case with most rollups on Ethereum, as they inherit the security of Ethereum itself and are thus trust-minimised. Furthermore, there's also the fabled 'decentralization' which can usually be measured in the cost of running a full node (which if, too expensive, will cause the decentralization of the network to falter). This is usually why we don't increase block sizes. Lastly, it must be permissionless to participate in the rollup's ecosystem, which is what is enabled by the rollups' bridge contract on Ethereum.

The underlying security is provided by the Layer 1, which provides data availability among others.

The Three Pillars of Fuel

The Fuel roadmap and concept are that of an execution layer that specializes in the future—a modular future. This is seen through their focus on specializing in providing the best possible execution layer for both Ethereum and other DA solutions. The three pillars of the Fuel design are:

1. Parallel transaction execution
2. The Fuel Virtual Machine (FuelVM)
3. A superior developer experience (with Sway and Forc)

Parallel Processing (Execution)

Parallel transaction processing was heavily popularised by Solana, through the Sealevel runtime which sorts incoming transactions to run in parallel across several cores which means they don't affect the same state in the VM's memory. This is because Solana transactions describe all the state transactions will read or write when it's being executed—similarly to UTXOs—thus enabling parallel processing. Solana is like Ethereum, account-based. However, unlike Ethereum, each node ensures that accounts accessed multiple times are only listed sequentially in one queue.

So how does a rollup like Fuel benefit from this? It does so since it's able to execute transactions in parallel by using the UTXO model. Therefore Fuel is able to use more threads and cores of a CPU, that in single-threaded blockchains are usually idle. As a result, Fuel can provide more compute and transaction throughput than other rollups on Ethereum.

FuelVM

The FuelVM is the virtual machine used on Fuel for building out various applications and smart contracts through the Sway language, which we'll get into in a bit. In the FuelVM, transactions happen through UTXOs, as covered earlier. Where you have inputs that get destroyed and outputs created. A virtual machine is basically a state computer that allows smart contracts to interact with one another. It also specifies the rules for changing the state of each Fuel block. If you're interested in how the FuelVM works and want to start building on Fuel, you can check out the GitHub, which has all the info you'll need as a developer.

Sway Programming Language

Sway is the language for building out smart contracts and applications on top of Fuel. It is heavily based on Rust, which is an already hugely popular language for building out blockchain applications (primarily used in the Cosmos ecosystem and on Solana and Near as well). Sway is optimised for use through the FuelVM and has a toolchain named Forc (the Fuel Orchestrator, pronounced "fork"). Forc provides tools and commands that developers can use within the FuelVM. You can compare Forc to Cargo, which is used as the Rust build system. This means that it is extremely easy for Rust developers to learn Sway, and start building on top of Fuel. Sway also has a vscode plugin, so it is extremely easy to get started with. Furthermore, Rust (of which Sway is a DSL of) has been voted the most loved programming language year after year by developers, and one of the most used as well.

L2 Tokenomics

To better understand how Fuel is trying to change the way rollups work currently, it is very important to look at their views on tokenomics for rollups. Tokenomics for rollups obviously work quite a bit differently, since you're still reliant on Ethereum for settlement, getting unified liquidity and inheriting the security. Fuel has discussed three major token models that should be avoided for rollups (but that might be fine for other systems such as sidechains). They are:

1. A Proof-of-Stake model where validators can censor new rollup blocks, which isn't needed since you inherit the security of Ethereum. A lot of rollups are trying to go this route with off-chain DA, such as ZKSync.
2. A model where you're required to use the native token to pay for fees, thereby tying it to Ethereum. However, the rollup nodes themselves are still required to hold Ethereum to settle. This just adds further UX hurdles for users using the protocol. An example of a rollup going this route is Obscuro.
3. A governance token that provides control over the rollup contract, which could be attacked via cryptoeconomic attacks.

So what model does Fuel see as the way forward?

The block space on the rollup is a limited resource, just like it is on Ethereum. However, the block space on the rollup is quite a bit different from Ethereum's. This execution space is scarce, just like Ethereum's, but it exists independently from it while still being correlated as you're dependent on the underlying layer's data availability. Fuel sees a future where this execution capacity can be tokenized, but not one where the token creates a higher fee or friction for the end-user. So how would you tokenize the block space?

You could tokenize the scarcity of block space, by giving token holders the right to collect fees as a block producer. This moves the token demand to block producers which have demand for being able to collect fees on future block space. In this way, the end-users can utilise whatever token that block producers want to accept for fees, as specified earlier in the article. Furthermore, by using a model in which you tokenize scarcity through the right to collect fees, you can decentralize the block production. This means that the node operators of the rollup will bond tokens, for the right to produce blocks and collect fees from end-users, which creates a marketplace for the token—but doesn't require the end-user to use that specific token. This means that the token is used for leader selection, the right to produce blocks and collect the fees associated with such. I have actually talked about this in regards to

what I would like to see other rollups on Ethereum do, but so far most of them have decided to go with the previous three models discussed further up.

MEV Capture

This capture of the value of block space also relates heavily to MEV, which is also a huge issue with centralized block producers. However, by moving towards a token model, as the one described, you're able to decentralize the operating nodes and their leader selection. This means a token model like this one, is also able to in a way 'tokenize' MEV, which is part of the value that the holders of the token derive the right to claim. By tokenizing the future cash flow of block space, the protocol is also tokenizing the future cash flow of ordering execution within that block space.

Other rollups are also doing fascinating research in regards to minimising MEV within their protocols. For instance, the auctioning off of MEV, which essentially allows you to tokenize it, has previously been discussed here by Karl Floesch from Optimism. Arbitrum and Chainlink are exploring a fair sequencing service (FSS), that should increase the fairness and predictability of transaction ordering on a rollup.

Fuel in a modular world

A modular world where most of the functionalities of an original monolithic blockchain are separated into several chains creates the possibility to optimize various parts of those layers. This is what Celestia is doing by optimising for Data Availability, and what Fuel is doing by optimising for Execution in a modular paradigm. This is also what enables the various layers to create optimised nodes, whether that is full, light or bridge nodes. These nodes are able to be optimized for their specific purpose, which should increase the capabilities of the various layers immensely.

Something I've talked about extensively over the various articles and threads I've done is the advance of state bloat, and how building modular can help alleviate some of that. This is what using a separate data availability layer that is stateless helps with strongly. Fuel thus seeks to go beyond the limitations of monolithic design principles.

Outside that, the transaction design is also extremely unique, since Fuel makes use of UTXO. This is quite unheard of since you're basically using 'old tech', but also one of the reasons why Fuel is so interesting to me.

Instead of building for the current climate on Ethereum, Fuel is building the engine for the autonomous future of modularity. This sets them apart from all current rollups. As I mentioned at the very beginning of the article, if you're interested in building on Fuel, learning Sway or similar, do attend the Fuel Hackathon and reach out to us—we'd love to talk to you.

References

https://fuel-labs.ghost.io/introducing-fuel-the-fastest-modular-execution-layer/

https://docs.fuel.sh/v1.1.0/Introduction/Welcome.html

https://github.com/fuellabs

https://forum.celestia.org/t/accounts-strict-access-lists-and-utxos/37

https://github.com/FuelLabs/fuel-specs/blob/master/specs/vm/main.md

https://fuel-labs.ghost.io/token-model-layer-2-block-production/

https://docs.fuel.sh/v1.1.0/Concepts/Fundamentals/Transaction%20Architecture.html

https://docs.fuel.sh/v1.1.0/Concepts/Fundamentals/Block%20Architecture.html

https://docs.fuel.sh/v1.1.0/Concepts/Fundamentals/Security%20Analysis.html#commondefinitions

https://fuellabs.github.io/sway/latest/

https://fuel.network/blog

Gustl May 19

How is this free? I love this sub

Reply Collapse

1 reply

Pinotio.com Writes Hype or Hodl · May 19

Thanks for this.

Could you clarify further (perhaps with an example) why fuel fees wouldn't end up having a dependence on Ethereum gas prices?

Reply Collapse

3 replies

4 more comments...