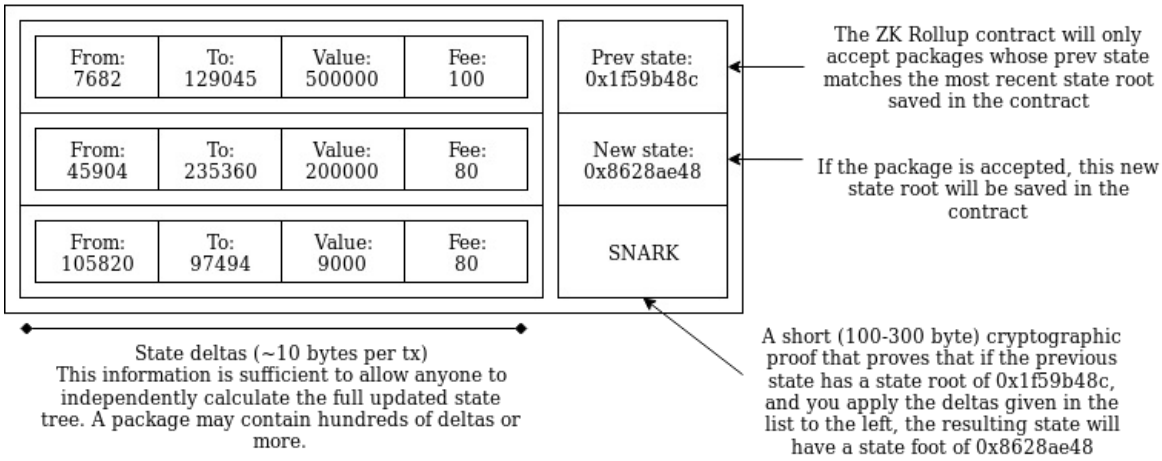
The Dawn of Hybrid Layer 2 Protocols

2019 Aug 28 See all posts

Special thanks to the Plasma Group team for review and feedback

Current approaches to layer 2 scaling - basically, Plasma and state channels - are increasingly moving from theory to practice, but at the same time it is becoming easier to see the inherent challenges in treating these techniques as a fully fledged scaling solution for Ethereum. Ethereum was arguably successful in large part because of its very easy developer experience: you write a program, publish the program, and anyone can interact with it. Designing a state channel or Plasma application, on the other hand, relies on a lot of explicit reasoning about incentives and application-specific development complexity. State channels work well for specific use cases such as repeated payments between the same two parties and two-player games (as successfully implemented in Celer), but more generalized usage is proving challenging. Plasma, particularly Plasma Cash, can work well for payments, but generalization similarly incurs challenges: even implementing a decentralized exchange requires clients to store much more history data, and generalizing to Ethereum-style smart contracts on Plasma seems extremely difficult.

But at the same time, there is a resurgence of a forgotten category of "semi-layer-2" protocols - a category which promises less extreme gains in scaling, but with the benefit of much easier generalization and more favorable security models. A long-forgotten blog post from 2014 introduced the idea of "shadow chains", an architecture where block data is published on-chain, but blocks are not *verified* by default. Rather, blocks are tentatively accepted, and only finalized after some period of time (eg. 2 weeks). During those 2 weeks, a tentatively accepted block can be challenged; only then is the block verified, and if the block proves to be invalid then the chain from that block on is reverted, and the original publisher's deposit is penalized. The contract does not keep track of the full state of the system; it only keeps track of the state root, and users themselves can calculate the state by processing the data submitted to the chain from start to head. A more recent proposal, ZK Rollup, does the same thing without challenge periods, by using ZK-SNARKs to verify blocks' validity.



Anatomy of a ZK Rollup package that is published on-chain. Hundreds of "internal transactions" that affect the state (ie. account balances) of the ZK Rollup system are compressed into a package that contains ~10 bytes per internal transaction that specifies the state transitions, plus a ~100-300 byte SNARK proving that the transitions are all valid.

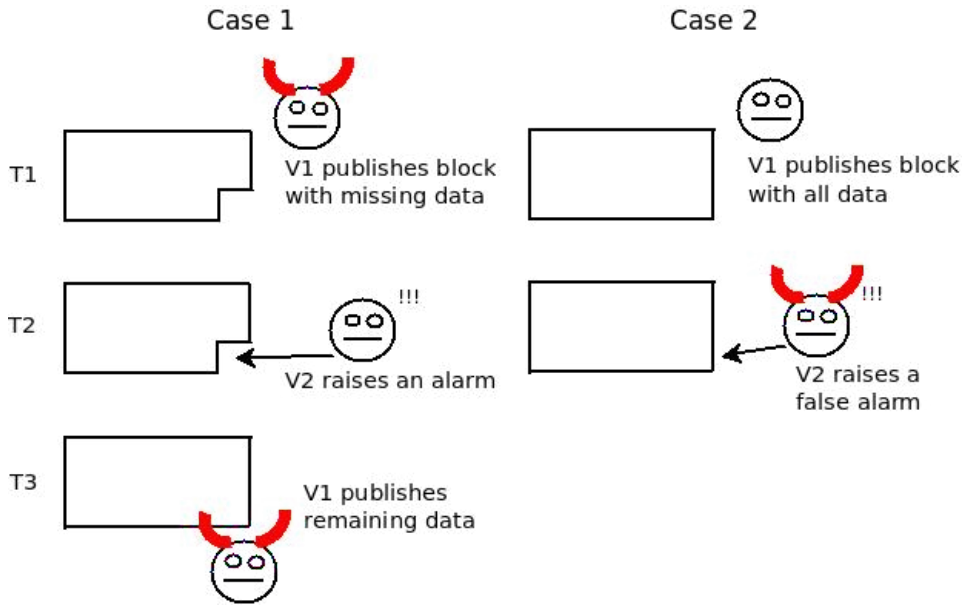
In both cases, the main chain is used to verify data *availability*, but does not (directly) verify block *validity* or perform any significant computation, unless challenges are made. This technique is thus not a jaw-droppingly huge scalability gain, because the on-chain data overhead eventually presents a bottleneck, but it is nevertheless a very significant one. Data is cheaper than computation, and there

are ways to compress transaction data very significantly, particularly because the great majority of data in a transaction is the signature and many signatures can be compressed into one through many forms of aggregation. ZK Rollup promises 500 tx/sec, a 30x gain over the Ethereum chain itself, by compressing each transaction to a mere ~10 bytes; signatures do not need to be included because their validity is verified by the zero-knowledge proof. With BLS aggregate signatures a similar throughput can be achieved in shadow chains (more recently called "optimistic rollup" to highlight its similarities to ZK Rollup). The upcoming Istanbul hard fork will reduce the gas cost of data from 68 per byte to 16 per byte, increasing the throughput of these techniques by another 4x (that's **over 2000 transactions per second**).

So what is the benefit of data on-chain techniques such as ZK/optimistic rollup versus data off-chain techniques such as Plasma? First of all, there is no need for semi-trusted operators. In ZK Rollup, because validity is verified by cryptographic proofs there is literally no way for a package submitter to be malicious (depending on the setup, a malicious submitter may cause the system to halt for a few seconds, but this is the most harm that can be done). In optimistic rollup, a malicious submitter can publish a bad block, but the next submitter will immediately challenge that block before publishing their own. In both ZK and optimistic rollup, enough data is published on chain to allow anyone to compute the complete internal state, simply by processing all of the submitted deltas in order, and there is no "data withholding attack" that can take this property away. Hence, becoming an operator can be fully permissionless; all that is needed is a security deposit (eg. 10 ETH) for anti-spam purposes.

Second, optimistic rollup particularly is vastly easier to generalize; the state transition function in an optimistic rollup system can be literally anything that can be computed within the gas limit of a single block (including the Merkle branches providing the parts of the state needed to verify the transition). ZK Rollup is theoretically generalizeable in the same way, though in practice making ZK SNARKs over general-purpose computation (such as EVM execution) is very difficult, at least for now. Third, optimistic rollup is much easier to build clients for, as there is less need for second-layer networking infrastructure; more can be done by just scanning the blockchain.

But where do these advantages come from? The answer lies in a highly technical issue known as the *data availability problem* (see note, video). Basically, there are two ways to try to cheat in a layer-2 system. The first is to publish invalid data to the blockchain. The second is to not publish data at all (eg. in Plasma, publishing the root hash of a new Plasma block to the main chain but without revealing the contents of the block to anyone). Published-but-invalid data is very easy to deal with, because once the data is published on-chain there are multiple ways to figure out unambiguously whether or not it's valid, and an invalid submission is unambiguously invalid so the submitter can be heavily penalized. Unavailable data, on the other hand, is much harder to deal with, because even though unavailability can be detected if challenged, one cannot reliably determine whose fault the non-publication is, especially if data is withheld by default and revealed on-demand only when some verification mechanism tries to verify its availability. This is illustrated in the "Fisherman's dilemma", which shows how a challenge-response game cannot distinguish between malicious submitters and malicious challengers:



Fisherman's dilemma. If you only start watching the given specific piece of data at time T3, you have no idea whether you are living in Case 1 or Case 2, and hence who is at fault.

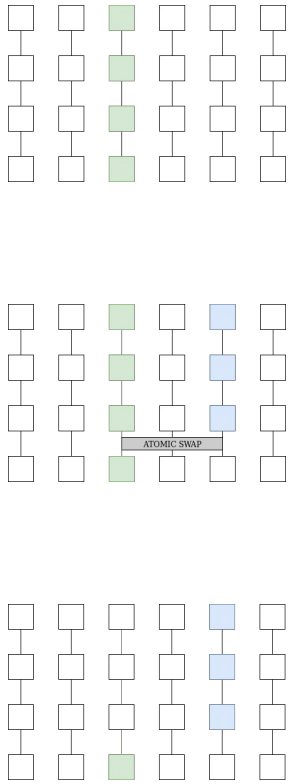
Plasma and channels both work around the fisherman's dilemma by pushing the problem to users: if you as a user decide that another user you are interacting with (a counterparty in a state channel, an operator in a Plasma chain) is not publishing data to you that they should be publishing, it's your responsibility to exit and move to a different counterparty/operator. The fact that you as a user have all of the *previous* data, and data about all of the transactions *you* signed, allows you to prove to the chain what assets you held inside the layer-2 protocol, and thus safely bring them out of the system. You prove the existence of a (previously agreed) operation that gave the asset to you, no one else can prove the existence of an operation approved by you that sent the asset to someone else, so you get the asset.

The technique is very elegant. However, it relies on a key assumption: that every state object has a logical "owner", and the state of the object cannot be changed without the owner's consent. This works well for UTXO-based payments (but not account-based payments, where you *can* edit someone else's balance *upward* without their consent; this is why account-based Plasma is so hard), and it can even be made to work for a decentralized exchange, but this "ownership" property is far from universal. Some applications, eg. Uniswap don't have a natural owner, and even in those applications that do, there are often multiple people that can legitimately make edits to the object. And there is no way to allow arbitrary third parties to exit an asset without introducing the possibility of denial-of-service (DoS) attacks, precisely because one cannot prove whether the publisher or submitter is at fault.

There are other issues peculiar to Plasma and channels individually. Channels do not allow off-chain transactions to users that are not already part of the channel (argument: suppose there existed a way to send $1 to an arbitrary new user from inside a channel. Then this technique could be used many times in parallel to send $1 to more users than there are funds in the system, already breaking its security guarantee). Plasma requires users to store large amounts of history data, which gets even bigger when different assets can be intertwined (eg. when an asset is transferred conditional on transfer of another asset, as happens in a decentralized exchange with a single-stage order book mechanism).

Because data-on-chain computation-off-chain layer 2 techniques don't have data availability issues, they have none of these weaknesses. ZK and optimistic rollup take great care to put enough data on chain to allow users to calculate the full state of the layer 2 system, ensuring that if any participant disappears a new one can trivially take their place. The only issue that they have is verifying computation without doing the computation on-chain, which is a much easier problem. And the scalability gains are significant: ~10 bytes per transaction in ZK Rollup, and a similar level of scalability can be achieved in optimistic rollup by using BLS aggregation to aggregate signatures. This corresponds to a theoretical maximum of ~500 transactions per second today, and over 2000 post-Istanbul.

But what if you want more scalability? Then there is a large middle ground between data-on-chain layer 2 and data-off-chain layer 2 protocols, with many hybrid approaches that give you some of the benefits of both. To give a simple example, the history storage blowup in a decentralized exchange implemented on Plasma Cash can be prevented by publishing a mapping of which orders are matched with which orders (that's less than 4 bytes per order) on chain:



***Left**: History data a Plasma Cash user needs to store if they own 1 coin. **Middle:** History data a Plasma Cash user needs to store if they own 1 coin that was exchanged with another coin using an atomic swap. **Right**: History data a Plasma Cash user needs to store if the order matching is published on chain.*

Even outside of the decentralized exchange context, the amount of history that users need to store in Plasma can be reduced by having the Plasma chain periodically publish some per-user data on-chain. One could also imagine a platform which works like Plasma in the case where some state *does* have a logical "owner" and works like ZK or optimistic rollup in the case where it does not. Plasma developers are already starting to work on these kinds of optimizations.

There is thus a strong case to be made for developers of layer 2 scalability solutions to move to be more willing to publish per-user data on-chain at least some of the time: it greatly increases ease of development, generality and security and reduces per-user load (eg. no need for users storing history data). The efficiency losses of doing so are also overstated: even in a fully off-chain layer-2 architecture, users depositing, withdrawing and moving between different counterparties and providers is going to be an inevitable and frequent occurrence, and so there will be a significant amount of per-user on-chain data regardless. The hybrid route opens the door to a relatively fast deployment of fully generalized Ethereum-style smart contracts inside a quasi-layer-2 architecture.

See also:

- Introducing the OVM
- Blog post by Karl Floersch

- Related ideas by John Adler