October 29, 2023

RPC3: A standard for decentralized remote procedure calls, leveraging IPFS and privacy-enabled blockchain

Mirayashi
mirayashi@proton.me

Abstract

RPC3 is not a protocol, but rather a standard that suggests a novel architecture for decentralized autonomous applications. IPFS and blockchain are both technologies enabling decentralization on the web, one for data storage and transmission, the other for consensus in code execution and exchange of value. The idea behind RPC3 is to leverage the best of both technologies in order to allow building multi-purpose and arbitrarily complex applications and online services that are truly owned and governed by their users. It basically works by storing application state and request payload data to IPFS before submitting their CID to a smart contract, which acts as a coordinator and decides the order in which requests are processed. Then anyone who has some computing resources and is willing to stake some value in the contract is able to collect the requests and execute them in their local environment, submitting the IPFS CIDs of the outputs of each request as well as the final state of the application back to the contract. The response is then delivered to the caller and rewards are distributed to network participants, if and only if the contract determines that a consensus has been established in the results. As it doesn't enforce any implementation details, apps built on top of RPC3 principles are free to add extra layers in order to achieve other goals, such as data privacy or integration with APIs of the web as we know today.

1 Introduction

Most online services nowadays are centralized on servers owned by companies motivated by profit. We are seeing a few emerging technologies aiming at bringing more decentralization to the web, a concept often referred to as *web3*. The vision of *web3* is to have online services working in an autonomous way that can resist to manipulation and censorship by companies and governments. Blockchain allows to execute code in a decentralized network that anyone with sufficient CPU power or value to stake can contribute to. Blockchain applications that are currently mainstream, known as *dApps*, are for most of them financial services that allow to exchange, lend or borrow value, which is indeed something we couldn't achieve in a decentralized manner without this technology. But when it comes to extend the usage of blockchain to different domains, such as supply chain, insurance, healthcare, calendars, word and image processing, social networks, or even games; as of today we still don't have any mainstream and truly decentralized app that cover the most everyday use cases.

2 Philosophy

If we take a deeper look at the solutions being developed today, it may help us understand the limitations that come into play when trying to push the scope of *dApps* beyond the exchange of financial value. Notably, we can observe that smart contract-based backends have the following properties:

- Computational resources of smart contracts are limited, whether it is CPU power, memory or disk storage. EVM chains for example use a gas system as a way to monetize these resources, and capacity depends on the maximum size of blocks which is very limited.
- Programming of smart contracts can only be done with a limited set of instructions. Because result needs to be deterministic and mathematically verifiable, we cannot use regular server-side programming languages, frameworks and libraries. We need to rely on oracles and inter-chain communication protocols to get data from external world, which adds layers of complexity.
- Deploying updates is a tough task as smart contracts are by design not upgradeable. Design patterns exist in order to mitigate this limitation, like the proxy pattern, but some cases may require more complex migration logic that inherently induces various risks.

These properties are not necessarily bad; in fact, they allow for maximal level of security, crucial for financial *dApps* to exist and operate. However, for use cases that don't manipulate financial value, let's say a todo list app or a social app, having such a high level of security is probably overkill. Moreover, such apps will certainly require more computational resources than a DEX or a NFT marketplace for example.

RPC3 philosophy is *off-chain-first*. Unlike ZK-based solutions [1] that require a prover and a verifier to execute off-chain code, RPC3 assumes that a result is correct as long as a majority of nodes agree that it is correct. If the majority agrees that 1 + 1 = 3, then the result will be accepted. The result is *not* mathematically verified. As such, security level is effectively lowered, while still being sufficient for non-financial use cases. And most importantly, this approach offers access to much cheaper computational resources.

3 Architecture

The general concept of RPC3 is that instead of one central server, we have many servers operated by independant individuals or organizations. All of these servers run the same off-chain application code, written with the usual programming languages, frameworks and libraries. A smart contract acts as a coordinator between all of these servers, while shared state storage and transmission of request/response payloads are done via IPFS.

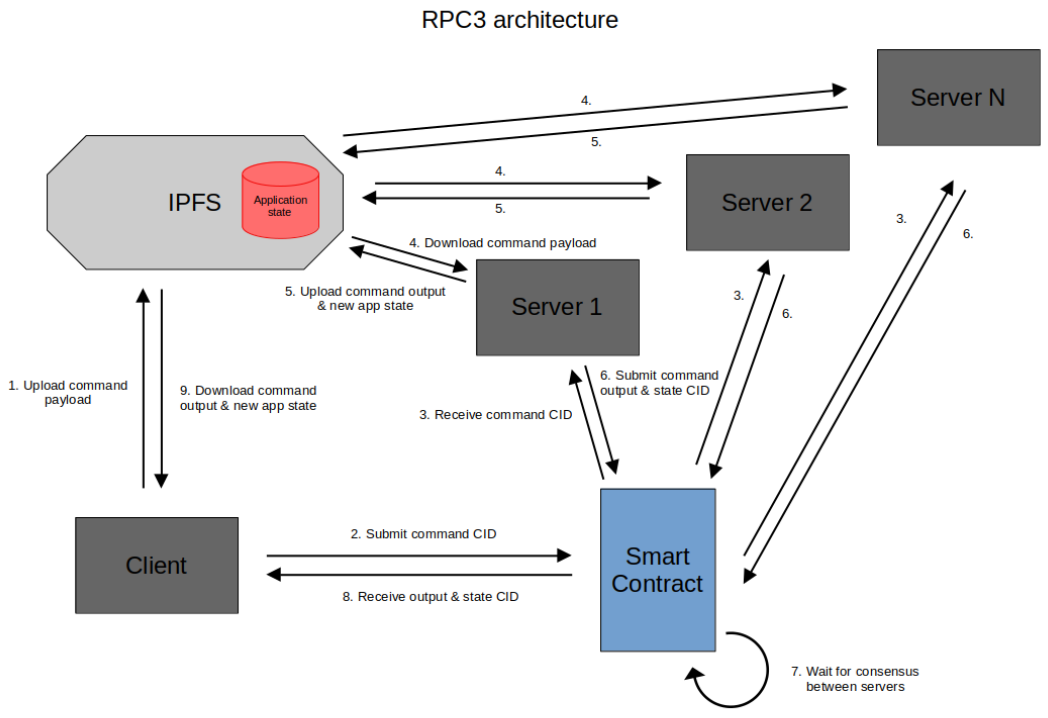
RPC3 architecture
IPFS
Application state
Server N
Server 2
Server 1
Client
Smart Contract
4.
5.
4.
5.
4. Download command payload
5. Upload command output & new app state
3.
6.
3.
6.
1. Upload command payload
9. Download command output & new app state
6. Submit command output & state CID
3. Receive command CID
2. Submit command CID
8. Receive output & state CID
7. Wait for consensus between servers

Figure 1: RPC3 architecture

- When the client wants to send a command, it uploads the payload (opcode, parameters, etc) of the command to IPFS, then submits the CID (IPFS hash-based content identifier) to the smart contract
- Servers are notified of incoming commands from the contract, they will download the content from IPFS and execute the command on their local environment
- Each server write the command output and the new application state to IPFS, and submit both resulting CIDs back to the contract
- The contract determines whether a consensus has been reached by comparing results between servers
- The client is notified that a response is available, from there they can get the CID and download the content from IPFS

4 Privacy of consensus

The consensus step requires the smart contract to be deployed on a blockchain that is able to keep the contract state private, such as Oasis Sapphire [2]. Indeed, the results submitted by servers must be kept private, otherwise one would be able to just blindly repost the results that are currently winning the consensus, instead of computing the results themselves. Although it doesn't prevent cases where many servers cooperate and ”share” results between each other (also known as Sybil attacks [3]), this can be mitigated for example by requiring servers to stake tokens in the contract to be able to participate.

5 Read-only queries

Sending commands is only useful if the client needs to perform an operation that modifies application state. For read-only queries, the client is expected to download the application state from IPFS and query the data themselves. This is a similar problem that we have with blockchains, where the client needs to download the whole blockchain by self-running a node in order to query data without relying on a third party. So just like blockchains, it would be totally possible to have intermediates that act as a gateway for clients to query data in a more convenient way.

6 Integration with external APIs

Since servers run code that is written in usual languages, they are technically allowed to call external web2 services through HTTP-based APIs for example. This gives developers the full responsibility not to create side-effects or rely on potentially non-deterministic sources. On the other hand, this has the huge benefit of removing the need for oracles. Bridging off-chain data to the on-chain world would be unnecessary, as everything runs off-chain by default.

7 Backend upgrades

Upgrading the code of servers is possible without redeploying the coordinator smart contract. The process would be similar to blockchain node upgrades, where a committee agrees on a block number that represents the deadline before which servers need to deploy the new code. If the application state needs a migration, launching the migration script would be a regular access-controlled command sent through the system just like any other command.

8 Incentive

An approach often adopted when it comes to design *dApps* for different fields than finance is to add financial incentives for the end user. We have seen the surge of *play-2-earn* games, where players are offered the opportunity to earn rewards while playing and monetize their progress. We can also mention *social-fi* that also involves incentivized participation from users of a social app. From my point of view, this concept is interesting as it does innovate new ways to engage a community and to build new kind of ecosystems and business models; however, it does not seem to bring any innovation to application architecture. In the end, blockchain here is only used to tokenize user-created content, but the servers that provide the utility to these tokens may still be centralized.

In RPC3-based apps, clients do not need to be incentivized in order to use the app; financial incentives exist only for those who run the backend. That way, business models would be similar to what we have today in web2 apps, such as ad revenue or subscriptions that the end users would pay for. The difference is that instead of going to companies wallets, income will be distributed among all backend operators according to their contributions.

9 User data privacy

RPC3 is a concept that is generic enough so it does not enforce any implementation details, including the format in which data should be transmitted. In the most simple case, we can imagine data being transmitted in JSON format without any encryption. But if let's say we want to build a private messaging app where messages should only be visible to chat participants, how would it be possible to achieve it so servers can process messages (e.g scanning them for spam detection) without them having knowledge of the content?

The two most common options are Zero-Knowledge Proofs (ZKP) [1] and Trusted Execution Environments (TEE) [4].

The former works well for basic scenarios, but is difficult to scale due to the complexity of the algorithms and the computationally expensive functions that it relies upon. The latter is cheaper and allows execution of arbitrarily complex code, but it requires servers to be equipped with compatible hardware and must properly implement remote attestation mechanisms for it to be secure.

The proposed solution is to leverage the privacy capabilities of Oasis Sapphire, as it actually utilizes TEEs under the hood. That way, we can use smart contracts deployed on Sapphire as a turnkey enclave solution to execute confidential functions. Servers won't need special hardware, all confidential operations would be effectively delegated to the contract. Note that the smart contract in question may be different than the one used to coordinate servers.

In our example, this contract would simply implement a function that accepts a ciphered message and returns whether the content is spam or not. For this to work, the contract must have knowledge of the decryption key, which implies that the client has called this contract beforehand to generate the key and encrypt the message. It is interesting to note that besides the encryption key, the contract would be completely stateless, which means that invoking its functions would not cost any gas.

10 Scalability

For the whole system to be viable, scalability is an important factor to consider. We should be able to estimate how many requests per second an application developed under this architecture may handle. Since Oasis Sapphire is the solution suggested for the consensus step, the scalability of the system will be mainly dependant on the scalability of the Sapphire blockchain itself:

- Block size is the same as Ethereum, which is of 30,000,000 gas units at the time this paper is written.
- Block frequency is one block every six seconds.
- One request is assumed to cost about 150,000 gas units.
- So that would represent about 200 requests per block, or 33 requests per second.

This assumes that we are using the full bandwidth of the blockchain, which is obviously not realistic. Fortunately, strategies are possible in order to improve this number significantly. For example, packing multiple requests into batches is the first solution that comes to my mind. But in this paper we will not dive too much into the different solutions that are possible to improve scalability.

It is also worth mentioning that a single request may take somewhere between 30 seconds and one minute to complete, because of the block time as well as the time required for consensus to be reached. This makes this architecture not suitable for applications requiring live data or frequent updates. The scope would be limited to *fast read, slow write* type of use cases.

11 Fault tolerance

Servers by definition may go down at anytime. It is important that the coordinator smart contract properly handle servers that don't actively and positively contribute to the consensus. Again, many solutions are possible here. The smart contract could implement a housekeeping logic that incentivizes active participants to punish the inactive ones by slashing their stake and excluding them from the next consensus rounds. Or we could have a more sophisticated reputation mechanism that will give more rewards and increase weight of active and honest servers.

In the end, the fault tolerance of the whole system mainly depends on the implementation of the coordinator smart contract.

12 Conclusion

Decentralized applications currently struggle to become mainstream in domains that don't involve the exchange of digital assets between end users. RPC3 is a honest attempt at proposing a solution to address the various limitations that may explain this observation. It is leveraging *web3* technologies that are already well established in order to model a new architecture for application backends, that hopefully will unlock new possibilities and allow the emergence of a new generation of online services truly governed by their users. The concept is very generic on purpose and only constitute a set of guidelines and suggestions. Actual implementations are welcome to enrich the specifications or add extra layers on top of them.

References

[1] Wikipedia. Zero-knowledge proof. https://en.wikipedia.org/wiki/Zero-knowledge_proof, 2023.

[2] Oasis Network. Sapphire. https://oasisprotocol.org/sapphire, 2023.

[3] Wikipedia. Sybil attack. https://en.wikipedia.org/wiki/Sybil_attack, 2023.

[4] Wikipedia. Trusted execution environment. https://en.wikipedia.org/wiki/Trusted_execution_environment, 2023.