Depermissioning Web3: a Permissionless Accountable RPC Protocol for Blockchain Networks

Weihong Wang
DistriNet, KU Leuven
Leuven, Belgium
weihong.wang@kuleuven.be

Tom Van Cutsem
DistriNet, KU Leuven
Leuven, Belgium
tom.vancutsem@kuleuven.be

***Abstract*—In blockchain networks, so-called "full nodes" serve data to and relay transactions from clients through an RPC interface. This serving layer enables integration of "Web3" data, stored on blockchains, with "Web2" mobile or web applications that cannot directly participate as peers in a blockchain network. In practice, the serving layer is dominated by a small number of centralized services ("node providers") that offer permissioned access to RPC endpoints. Clients register with these providers because they offer reliable and convenient access to blockchain data: operating a full node themselves requires significant computational and storage resources, and public (permissionless) RPC nodes lack financial incentives to serve large numbers of clients with consistent performance.**

Permissioned access to an otherwise permissionless blockchain network raises concerns regarding the privacy, integrity, and availability of data access. To address this, we propose a Permissionless Accountable RPC Protocol (PARP). It enables clients and full nodes to interact pseudonymously while keeping both parties accountable. PARP leverages "light client" schemes for essential data integrity checks, combined with fraud proofs, to keep full nodes honest and accountable. It integrates payment channels to facilitate micro-payments, holding clients accountable for the resources they consume and providing an economic incentive for full nodes to serve. Our prototype implementation for Ethereum demonstrates the feasibility of PARP, and we quantify its overhead compared to the base RPC protocol.

***Index Terms*—Blockchain Networks, Node-as-a-Service, Light Client, Payment Channel, RPC Protocol, Verifiable Data Access**

I. Introduction

Blockchain networks support decentralized applications or "dApps" by storing data on a peer-to-peer (P2P) network, as opposed to traditional "Web2" applications supported by centralized server-centric data storage. This shift introduces complexities for application clients because participating directly in a blockchain network's P2P protocol is challenging. Not only does it require the client to act as a server, but it also requires the client to potentially maintain a full copy of the network state. However, the computational and storage demands of running a so-called "full node" are considerable (e.g., Ethereum nodes require 2TB of SSD storage and 25 MBit/s bandwidth [1]). Clearly, it is not practical for all end-users to set up such infrastructure.

Additionally, due to the resource-intensiveness of operating a full node, combined with the lack of any economic incentives, the owners of those full nodes are hesitant to provide end-users with free and unrestricted access to blockchain data through their nodes' RPC interfaces, especially as the demand and number of application clients grows larger.

Consequently, many resource-limited clients rent access to full nodes through node-as-a-service (NaaS) [2] providers or simply "node providers". Popular node providers include Infura [3] and Alchemy [4], which operate full nodes on several blockchain networks and offer hosted RPC services. The NaaS trend emerged as a convenient solution for users to interact with Web3 protocols. Figure 1 illustrates a typical scenario where a dApp connects to a node provider, which relays user transactions to the blockchain network and retrieves data from the blockchain for presentation to the user.

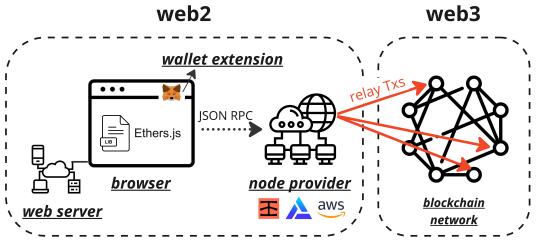

Fig. 1: A schematic of the typical serving layer underlying decentralized applications (dApps) using Web3 protocols.

The connection between applications and nodes run by node providers involves an API key, which authenticates who the user of this key is. By tracking and controlling how the API is used, node providers can charge the user based on their pricing plans. In the majority of cases, applying for an API key requires users to undergo a registration process, which may include sharing sensitive information such as personal identification or financial information.

The reliance on a trusted intermediary to interact with a trustless, decentralized network runs counter to the core principles of why Web3 protocols were invented in the first place [5]. In particular, users often trade **privacy**, **integrity**, and **availability** for convenience, much like they did in "Web2". Specifically:

- **Privacy.** When registering at a NaaS provider, users have to authorize the collection of personally identifiable information (PII), sometimes including their full name and credit card details, making it easy to profile users.

- **Integrity.** Node providers can manipulate or misrepresent blockchain data without penalties [5], leaving users without hard guarantees that the retrieved data is in accord with the latest data stored on the blockchain.
- **Availability.** Node providers face regulatory implications from laws in their operating countries, potentially leading to censorship of sanctioned applications and users in specific regions [6].

The blockchain RPC serving layer lacks accountability and economic incentives to provide applications and end-users with reliable access to on-chain data. In Ethereum, for example, full nodes receive financial rewards for validating transactions using "Proof-of-Stake" consensus and by charging fees on validated transactions, but not for serving RPC requests or relaying transactions. Misbehavior as a validator (e.g., by attesting to an incorrect block in Ethereum) is discouraged through slashing conditions, but no such mechanism exists for misbehavior in the serving layer. Yet, both validation and serving play a crucial role in a healthy network.

In short, Web3 protocols heavily rely on economic incentives to keep nodes honest and participatory, while such incentives are missing for the serving layer. We aim to address this gap by introducing mechanisms in the serving layer that enable (proportional) financial rewards for nodes that serve and, at the same time, ensure detection and punishment of misbehavior towards clients.

To introduce these mechanisms, we designed PARP, a **P**ermissionless **A**ccountable **R**PC **P**rotocol that wraps a blockchain network's base layer RPC protocol with additional layers of authentication and accounting. PARP aims to enhance the end users' access to reliable blockchain data while at the same time compensating full nodes fairly for the service provided. PARP leverages known light client schemes to improve data integrity for the clients. In addition, it introduces payment channels between clients and full nodes to allow full nodes to accept micro-payments from clients while safeguarding the pseudonymity of both parties. To act as a PARP server and accept such micro-payments, a full node must lock up funds that can be slashed on misbehavior, thus deterring malicious actors in the serving layer.

Our main contributions are the following:

- We propose a new serving layer model that complements the current endpoint infrastructure provided by permissioned services (NaaS providers). It features accountability mechanisms and incentives that mutually benefit clients and full nodes.
- We propose a design that safeguards the end users data security and does not require trust in a centralized third party.
- We implement PARP for the Ethereum RPC layer and integrate it into a widely used Ethereum execution node implementation (Geth), providing evidence for the feasibility of the protocol and its compatibility with current blockchain networks.

II. Web3 Serving Layer Issues

In this section, we discuss key considerations in the Web3 serving layer, including insufficient economic incentives for full nodes, dominance of major node providers and how their permissioned registration processes enable extensive profiling, and the trade-offs between accountability and permissionlessness in accessing blockchain networks.

A. Lack of Economic Incentives for Full Nodes

As light clients emerge to address heavy resource requirements for devices like smartphones, they face the challenge of bootstrapping and synchronizing themselves securely and efficiently [10]. However, serving requests from many light clients places a substantial burden on these full nodes regarding network resources and communication. Therefore, with an increasing number of light clients benefiting from enhanced privacy and security joining the network, the establishment of effective incentives to compensate full nodes for supporting light clients becomes crucial.

B. Centralization among Node Providers

To evaluate the role of node providers, we analyzed a dataset [11] from Torres *et al.* [12], which contains detailed records of web traffic from 1572 dApps. From this, we specifically focused on 383 dApps that send JSON-RPC calls directly from their frontend to node providers for blockchain data access. We then mapped these JSON-RPC calls to identify which node providers each dApp interacts with, noting that a single dApp can connect to multiple providers. This allows us to determine the extent of traffic received by each provider. The results indicate that **47.52%** of the dApps in our dataset connect to Infura [3], making it the most widely used provider by a significant margin. Alchemy [4] is the second-most common, used by **31.07%** of dApps, followed by Binance (12.01%), Ankr (9.4%), Cloudflare (6.79%) and other providers.

C. Permissioned Nature of Node Providers

We picked five top node providers from our dataset, excluding network-specific ones. We examined their traits on their websites during the API key application process, as shown in Table I.

First, we inspected the **registration requirements** of each provider to see if their services could be used without registration. Ankr stands out as the only provider that has a list of free endpoints.

For **wallet-based identity**, only Ankr supports this, granting users flexibility in service access. Conversely, the remaining providers ask for at least an email address for registration.

We also assessed **free monthly requests** and the pricing models of each provider. All providers offer some free daily service to users, while 3 out of 5 charge based on varied call types for a fairer fee calculation.

Finally, we reviewed the accepted **payment methods**, finding that 2 out of 5 providers accept crypto.

In conclusion, end users still need to register for an account to access a reliable blockchain connection, much like in Web2

TABLE I: Comparison of Features and Registration Requirements of Node Providers[‡]

Node Provider	Free Public Service	Login	Sign-up				Pricing Plan		Freemium Node Service	Payment		Traffic Share
	No Signup	Via wallets	Email	Full name	Org name		Call-Based	Plan tiers	Free usage[†]	Credit card	Crypto	
Infura [3]	-	-	●	●	-		-	5	3 million credits (daily)	●	-	182/383 (47.52%)
Alchemy [4]	-	-	●	●	-		●	4	300 million compute units (monthly)	●	-	119/383 (31.07%)
Ankr [7]	●	●[*]	◐	-	-		-	4	30 requests (per sec)	●	●	36/383 (9.4%)
Quicknode [8]	-	-	●	●	●		●	5	10 million API credits (monthly)	●	-	16/383 (4.18%)
Chainstack [9]	-	-	●	●	●		●	4	3 million request units (monthly)	●	●	5/383 (1.31%)

● = indicates that the property is provided; in the Sign-up column, it means the property is required.;
◐ = indicates that property is not necessarily required; - = does not provide property.

[*] Wallets must have active transactions in the past to be supported by the node provider. [‡] All the data was collected before December 2024.

[†] These metrics (compute units, requests, credits, etc.) represent the amount of computational work and resource usage as defined by different node providers.

services. In combination with all the Web3 requests sent to these node providers, it becomes fairly easy for them to construct user profiles.

D. Tradeoffs Between Accountability and Permissionlessness

Today clients can access some public RPC endpoints, either from node providers or anonymous full nodes [13], [14], but each option comes with its own tradeoffs. For node providers, access is permissioned due to registration requirements, but users can put some trust in the data as node providers have a reputation and commercial interests to uphold. By contrast, public anonymous RPC endpoints are permissionless, but there is no accountability for these anonymous nodes to serve information reliably and correctly. Hence, achieving both permissionlessness and accountability in the serving layer remains an open issue.

III. Background

In this section, we discuss the state of practice in bridging Web2 and Web3 protocols. We also provide the necessary background on light client schemes and payment channels.

A. Endpoint Infrastructure

To interact with the blockchain network, clients connect to a full node via universally supported APIs like JSON-RPC. An RPC endpoint is the network location where an application sends its API requests to a full node for execution. The endpoint typically provides access to various functionalities of the blockchain network, including broadcasting transactions and retrieving blocks and block headers.

Most dApp developers obtain an API key with a node provider for the network they wish to use (as discussed in the Introduction I). DApp end-users often can't configure the underlying API keys that their dApps are using. Although many wallets do allow end-users to configure their own RPC URL, most opt to use the default settings, commonly referring to a node provider url. For example, MetaMask, a widely-used Web3 wallet for Ethereum uses Infura [15] as its default endpoint provider to query the Ethereum blockchain and obtain the balance for the end-user's addresses.

B. Light Client Solutions

Light client schemes, such as simple payment verification (SPV) [16], offer a compromise between the resource-intensive demands of operating a full node and the security risks of relying on third-party servers.

Efficiency. Light clients download only block headers, significantly reducing data requirements (e.g., an 80-byte header vs. a 1MB full block). Schemes, such as FlyClient [17] and Coda [18], achieve fast block header synchronization even up to a constant size to the length of the chain.

Data integrity. By leveraging Merkle trees, light clients can verify transactions or information against the transaction root or state root of the tree contained in the block header.

Reliance on full nodes. Due to storage limitations of light clients, they still rely on their peer full nodes to obtain essential information to keep up-to-date with the tip of the chain.

C. Payment channels

Payment channels were proposed to enable micropayments between two parties without recording each individual payment as a transaction on the blockchain [19], thus **increasing throughput** and **reducing costs** (fees).

A payment channel operates under predefined rules set by a smart contract [20]. It is opened with one on-chain transaction in which both parties deposit funds, indicating the total amount available for "off-chain" transactions. The parties then manage a local off-chain ledger to track their balances. Upon closing the channel, if both agree on the final balance, funds are settled accordingly on the blockchain. In case of disputes, the smart contract acts as a mediator, verifying the provided evidence and disbursing funds based on the validated final state.

IV. PARP Protocol Overview

We describe the design goals of PARP, introduce terminology and assumptions specific to our protocol, and then describe the full lifecycle of a PARP connection.

A. Overview and Design Goals

PARP is an RPC protocol designed to allow a *light client* to interact with a blockchain through a *full node* in such a

way that the interaction is both *permissionless*, *accountable* and *economically sustainable* for both parties.

To participate as a full node in PARP, the node's operator must first deposit tokens as collateral to incentivize honest behaviour. While alternative approaches to accountability exist, PARP adopts a collateral-based mechanism to provide clear and quantifiable assurances for both parties.

To receive service from a PARP full node, a PARP light client must first open a payment channel and commit to deposit funds in the channel before it can start making RPC requests. These funds represent the light client's budget to pay the full node for its service. Additionally, the PARP protocol allows the light client to detect incorrect RPC responses and report them (via another full node) through a fraud-proof protocol, penalizing the misbehaving PARP full node (by withholding part of its collateral).

The design goals of PARP include:

1) Accountability:
   - **Bilateral assurances.** Deposits by light clients and full nodes establish mutual accountability, ensuring reliable services and fair compensation within the protocol.
   - **Trust and verification.** By leveraging merkle proofs, light clients can verify the correctness of the data returned in an RPC response. Together with a fraud-proof protocol, the full node is incentivized to serve RPC requests correctly.
2) Permissionlessness:
   - **Pseudonymity.** Our protocol reduces the leakage of personally identifying information by enabling end-users and dApps to interact with blockchains without requiring registration and without API keys that can correlate requests and build up user profiles.
   - **Enhanced availability.** PARP full nodes are discoverable via an on-chain registry and because of the lack of any sign-up process, clients can trivially switch between different PARP full nodes, e.g., for fail-over.
3) Economic incentives:
   - **Cost-efficient payments.** Micropayments in a payment channel are used to compensate full nodes for their services, incentivizing them to serve light clients. The funds deposited by a light client are specifically allocated for payments to a full node within the channel.

B. Roles

The protocol involves three main roles:

Full Node: A computer running the PARP-compatible full node software and connecting to other (PARP or non-PARP compatible) full node peers. It stores complete blockchain data, though in practice, it may be periodically pruned to save disk space [21]. To serve as a full node, it must deposit tokens to a PARP-specific smart contract as collateral.

Light Client: A computer with limited storage, compute and network resources. A light client downloads only **block headers** rather than full blocks. It interacts with its connected PARP full node by making RPC requests, manages essential chain information, verifies data integrity using Merkle proofs against the root hash in block headers. It generates fraud-proofs if it detects invalid RPC responses.

Payment Channel: A unidirectional ledger payment channel set up between a light client and a full node, signifying a successful connection between the two parties. A light client must deposit funds into this channel on-chain, representing the total amount of tokens it will pay a full node for its services.

C. On-Chain Modules

The protocol also includes three additional on-chain modules deployed as smart contracts:

Full Nodes Deposit Module: This module enables a full node to deposit its tokens, making it eligible to serve light clients in the network.

Channels Management Module: This module manages payment channels for each PARP connection. It is responsible for managing the state of the payment channels, including the allocation of funds between the participating light client and the full node upon connection closure. Each payment channel has a unique identifier, based on the identity of the participants, which is recorded in this module.

Fraud Detection Module: This module processes fraud proofs from light clients. If verified, it penalizes misbehaving full nodes by slashing their collateral and rewards the reporting participants, as detailed in Section IV-F.

Figure 2 illustrates how all the participants in the PARP protocol interact.

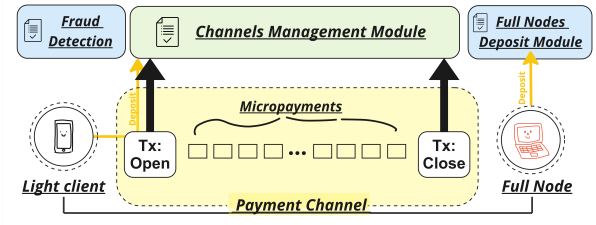

Fig. 2: Primary participants in a PARP connection.

D. Assumptions

A light client relies on trusted sources to remain updated with the latest block headers in the blockchain [16]–[18]. We assume that the light client can request and receive **block header**, which is compact, not client-specific, and serves as the root of trust, from *any* full node (PARP-compatible or not), without payment. Access to block headers enables the light client to independently verify data integrity and network liveness.

Furthermore, our protocol assumes a **strong synchrony** model. Specifically, we assume that messages between honest parties are delivered within a bounded delay.

E. Settings and Lifecycle of a PARP Connection

*1) **Bootstrapping**:* Before initiating a PARP connection, the light client retrieves block headers from full nodes summarizing the blockchain state, including the chain tip and

relevant Merkle roots. It then handshakes with a selected PARP-compatible full node to agree on connection parameters.

2) ***Connection Setup****:* The connection is created through an on-chain transaction to open a payment channel. This transaction is authorized by the light client and mediated via the full node, and includes the transfer of a client deposit, and the (pseudonymous) identity of both participants.

3) ***Request and Response phase****:* Once the payment channel is created, light client and full node can transact "off-chain". Light clients can send any number of *Request* messages, and full nodes respond with *Response* messages.

When a light client initiates a Request, it must include a micropayment, which indicates the payment amount, along with a blockchain-specific RPC call (e.g., for Ethereum, a call like *eth_getBalance(address)*), and other parameters. The payment amount is cumulative, meaning it adds up the total amount owed by the light client for all previous calls along with the current call. Subsequently, the full node responds to the request, and both parties retain relevant records. Both the light client and the full node primarily track the requests, as each request contains a signed cumulative payment amount that enables the full node to redeem these funds. The response, recorded by the light client, may be used to generate a fraud-proof, as discussed in Section IV-F.

4) ***Closure (with or without dispute)****:* The connection closure can be triggered by either party through another on-chain transaction. The transaction must include the latest signed payment amount to close the channel. The channel will have a dispute window for a period of time before it closes.

Closure without dispute. The final state of the channel is settled on-chain. The funds are distributed accordingly: the remaining unspent budget is returned to the light client, and the full node receives payment. Once this process is completed, the channel is settled, and the connection is closed.

Closure with dispute. Either party can present the latest state of the channel, represented by a payment proof (a signed message from the light client) with the largest cumulative payment amount. After the dispute period ends and the dispute is resolved based on the validity of the submitted payment proof, the channel is settled, and the connection is closed.

F. Fraud-Proof Protocol

The light client verifies the response from the full node by applying several checks. Based on the checks, the response is treated as:

- **Valid**: All checks pass successfully; the client trusts it.
- **Invalid**: The client cannot trust the response, but also cannot hold the full node accountable for fraud. It is sensible for the client to terminate the connection.
- **Fraudulent**: The client cannot trust the response, terminates the connection, and can take steps to construct a fraud proof to penalize the full node for its misbehavior.

In the case of fraud, to penalize the full node, the client must submit a fraud proof to the *Fraud Detection Module* contract. Obviously we cannot trust the full node to submit a proof of its own fraudulent behavior to the blockchain. The light client must instead resort to another PARP-compatible full node, which we refer to as a **witness full node**.

To verify the fraud proof, the *Fraud Detection* contract can use the request and response data to re-check all the conditions stated above.

If the fraud proof is deemed valid by the *Fraud Detection Module*, the contract will instruct the *Deposit Module* to confiscate the deposit of the full node and distribute it to three parties: the network's serving layer nodes (to incentivize punishment of fraudulent nodes), the light client (to incentivize reporting of fraudulent nodes), and the witness full node (to incentivize assistance in reporting fraudulent nodes). The witness node is compensated by the contract directly. The light client does not need to establish a payment channel with the witness node.

V. PARP Design Details

This section outlines the network assumptions, introduces the protocol states and messages, and further illustrates the state transition within participants in a PARP connection. It also discusses the liveness check on payment channels and fraud detection mechanisms.

A. Protocol States and Messages

A full node is denoted as FN, a light client as LC, and a payment channel as $\mathcal{P}$. Let (pk_{FN}, sk_{FN}) and (pk_{LC}, sk_{LC}) denote their public/private key pairs, respectively, with $addr$ denoting their corresponding addresses.

Within the off-chain connection, a light client's request to a full node is denoted as req, while a corresponding response is represented as res. Both the request and response in each RPC call round i must be signed by their respective parties before being sent. The data structure is defined in Figure 3.

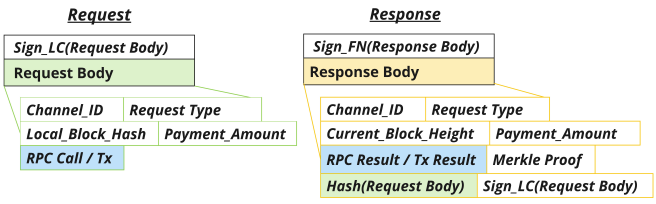

Fig. 3: Structure of a PARP request and a PARP response

Let the request be denoted by req and defined as follows:

$$req = (\alpha, h_B, a, \gamma, h_{req}, \sigma_a, \sigma_{req})$$

where:

- α is the identifier of the channel between LC and FN.
- h_B is the most recent valid hash of a blockchain block, denoted as B, stored by LC.
- a indicates the amount LC is willing to pay for all previous calls along with the current call. It should satisfy $req_i.a \geqslant req_{i-1}.a$, indicating cumulative payments.
- γ is an RPC call that LC wants to get executed.
- h_{req} is the result of $Hash(\alpha, h_B, a, \gamma)$.

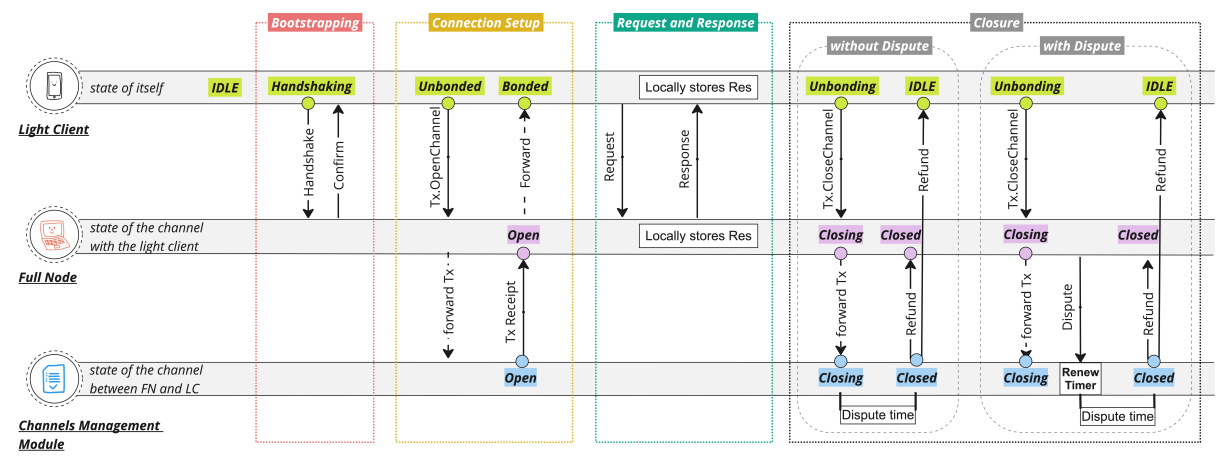

Fig. 4: Tripartite state transition diagram illustrating the lifecycle of a PARP connection.

- σ_a is $Sign(Hash(\alpha, a), sk_{LC})$, for payments.
- σ_{req} is $Sign(h_{req}, sk_{LC})$, for verification.

And the response is denoted as res:

$$res = (\alpha, m_B, a, R(\gamma), \pi_\gamma, h_{req}, \sigma_{req}, \sigma_{res})$$

where:

- α is the identifier of the channel between LC and FN.
- m_B is the current block height, indicating the time of response and also specifies which block header should be checked for π_γ. It has to be greater than or equal to the block height indicated in the request by $req_i.h_B$.
- a needs to match the input value $req.a$ for h_{req}.
- $R(\gamma)$ is the result for $req.\gamma$, if applicable.
- π_γ is the Merkle Proof of Inclusion of $R(\gamma)$.
- σ_{res} is the result of $Sign(h_{res}, sk_{FN})$, where

$$h_{res} = Hash(\alpha, m_B, a, R(\gamma), \pi_\gamma, h_{req}, \sigma_{req})$$

A unidirectional payment channel stored in the Channels Management Module (CMM) on the blockchain is denoted as $\mathcal{P}$:

$$\mathcal{P} = (\alpha, LC, FN, b, cs, T)$$

where:

- α is the unique identifier for the payment channel.
- LC is the address of the light client.
- FN is the address of the full node.
- b is the budget locked in this channel by LC, limiting the maximum allowable $req.a$.
- cs represents the latest state of the payment channel, submitted by either participant and must be validated by a correct a and σ_a to be accepted.
- T denotes the status of the payment channel, with three possible values: $Open$, $Closing$, $Closed$.

In our scenario, LC only manages one payment channel $\mathcal{P}$ locally, while FN and the CMM oversee several payment channels using identifiers with a mapping $(\alpha \mapsto \mathcal{P})$.

The channel state of a $\mathcal{P}$ stored locally by LC and FN are the values of α, a and σ_a exchanged in each round. Following the settlement of the channel, where the final channel state will be submitted on-chain, the funds are redistributed to the participants based on the payment amounts owed by LC to FN. Moreover, res serves as a part of a fraud-proof sent by LC to the CMM to detect inconsistent returned data from FN.

B. State Transition of Participants

The state transition diagram depicting the entire lifecycle of a PARP connection is illustrated in Figure 4.

A PARP-compatible full node can either be available or not available to a light client's payment channel connection request. To become available, a full node must deposit funds to the FNDM and indicate they are ready to serve.

The state of a light client LC is deemed $IDLE$ if there is no established connection with a full node. It begins with the $Handshaking$ state to the $Unbonded$ state and subsequently to the $Bonded$ state upon establishing a payment channel. Later, it may enter the $Unbonding$ state if the connection is ending, ultimately returning to the $IDLE$ state.

The state of a payment channel $\mathcal{P}$ can be classified into three states: $Open$ means it is successfully set up; $Closing$ means one party wants to settle the channel, the channel will be under a period of time for disputes where the fund is time-locked; $Closed$ means it is successfully settled, and both parties receive the correct balance from the channel back.

1) **Initialization**: During this phase, LC seeks consent from FN to establish the payment channel $\mathcal{P}$. In our design, where a full node is not required to deposit funds into a payment channel, mutual consent between LC and FN is crucial for channel creation. This confirmation includes an expiry time, indicating when the confirmation will expire, requiring LC to initiate another handshake if necessary. The algorithm is explained in Algorithm 1.
2) **Channel Opening**: The creation of a payment channel $\mathcal{P}$ is initiated by an $OpenChannel$ transaction sent from LC, transitioning LC's state to $Unbonded$. This transaction includes metadata such as the budget amount $\mathcal{P}.b$, participant addresses $\mathcal{P}.LC$ and $\mathcal{P}.FN$, and the

Algorithm 1 Light Client State Transition Logic during Initialization and Channel Opening Stages

Initialization:
  Generate (pk_{LC}, sk_{LC}) and its address denoted as LC
  DECLARE h_B: a block HASH
  DECLARE FN: an ADDRESS that represents a full node FN
  DECLARE $step$: one of the values in (IDLE, Handshaking, Unbonded, Bonded, Unbonding)
  DECLARE α: an INTEGER to identify a payment channel
  DECLARE a: an INTEGER to record the usage of the budget, representing the latest local state of a channel
upon start **call** StartHandShaking()
function STARTHANDSHAKING
  $h_B \leftarrow$ Fetch the latest block hash from the network
  $FN \leftarrow$ Pick a full node
  Send msg$\langle$HANDSHAKE, $LC\rangle$ to FN
  $step \leftarrow$ Handshaking
  Set the $hsTimer$ timer
end function
upon receive msg$\langle$HSCONFIRM, pk_{FN}, $expiryDate$, $Sign((LC||expiryDate), sk_{FN})\rangle$from FN while $hsTimer$ is active and $step = Handshaking$
  Verify $Sign((LC||expiryDate), sk_{FN})$ with pk_{FN}
  return if signature is not valid
  Form and Sign the Tx$\langle$OpenChannel($Sign((LC||expiryDate), sk_{FN})$, pk_{FN}, pk_{LC}, $expiryDate\rangle$
  Attach the budget to Tx$\langle$OpenChannel$\rangle$
  Send Tx$\langle$OpenChannel$\rangle$ to FN
  $step \leftarrow$ Unbonded
upon receive TxReciept$\langle$OpenChannel, $Sign(channelId, sk_{FN}), channelId\rangle$ from FN while $step = Unbonded$
  Verify $Sign(channelId, sk_{FN})$ with pk_{FN}
  $\alpha \leftarrow channelId$
  $a \leftarrow 0$
  $step \leftarrow$ Bonded

signed confirmation from FN. Additionally, it involves the transfer from LC of a certain amount of money equal to $\mathcal{P}.b$. Upon receiving the $TxReceipt$ of the $OpenChannel$ transaction, LC transitions to the $Bonded$ state, and the channel's state managed by FN and CMM is set to be $Open$, with the identifier α assigned.

3) **Active Phase**: This phase does not involve any on-chain participation. LC generates a req and sends it to FN. FN then responds with a res and stores $req.a$ and $req.\sigma_a$. LC stores the $req.a$ locally and verifies res.
4) **Termination**: Either party can send a $CloseChannel$ transaction which includes (α, a, σ_a) to the network. The action transitions LC to the $Unbonding$ state and $\mathcal{P}$ to the $Closing$ state.
   - **No dispute.** If there is no dispute, which is the ideal case, after a period of dispute time, the channel is officially and successfully closed on-chain. CMM distributes $\mathcal{P}.b$ accordingly based on a. The channel state managed by FN proceeds to $Closed$, while LC returns to $IDLE$.
   - **Dispute present.** Before the closure of the channel, either party can submit a final state different from the $\mathcal{P}.cs$ recorded by CMM. The valid state with a higher value of a will be acknowledged as the most recent state. Whenever a party submits a new valid latest state, the dispute time will be reset to allow the other party enough time to respond. The rest is the same as for the no-dispute scenario.

C. Liveness Check on the Payment Channel

To facilitate a light client to monitor the payment channel's liveness, for example, if the payment channel is closed secretly by a full node, LC periodically sends a request to FN asking for $\mathcal{P}.T$. By getting block header information from other sources in the network as described in Section IV-D, a light client can verify the liveness of a channel.

D. Fraud Detection and Reporting

To ensure system integrity, both the light client and the on-chain module perform a series of verification steps. The light client serves as the first line of defense, performing local checks to classify responses as **valid**, **invalid**, or **fraudulent**. If fraud is detected, it submits the relevant information as a fraud proof to the on-chain Fraud Detection Module, which then independently verifies the proof and enforces penalties.

The light client verifies the response from the full node through the following checks:

- ***Verify Request Hash***: To cryptographically link a PARP request with its response (which is needed to establish a

fraud proof), the response must include the hash of its associated request (*request hash*). The client must verify that the request hash matches the expected one. If not, the response is classified as **invalid** because the light client would not be able to generate a fraud proof, thus it cannot trust the response. If the hash matches but the request's signature fails verification, the response is also invalid.

- ***Verify Response Signature***: The response message must contain a valid signature from the full node. If the signature is invalid, then again the client would not be able to generate a fraud proof to hold the full node accountable, so the response is classified as **invalid**.
- ***Channel Identifier Check***: The channel ID in the response must match the one in the request. Any mismatch is classified as **invalid**.
- ***Payment Amount Check***: The payment amount in the response must match the cumulative amount signed by the light client in the request. Any mismatch is **fraud**.
- ***Timestamp Check***: The block height in the response must not be lower than the one indicated in the request by the block hash. Any mismatch is considered **fraud**.
- ***Verify Merkle Proof***: The response must contain a Merkle proof showing the data is part of the tree specified by the request type (e.g., transaction trie or state trie) at the current block height in the response. If the proof fails to verify, the response is considered **fraud**.

When the light client detects fraud, it submits the request req, the response res, and the address of the witness node $addr_{WN}$ to the on-chain module.

The on-chain module requires a trusted root hash h_{root} for the relevant Merkle tree, determined by the return block height $res.m_B$. It will conduct the following checks:

The integrity of the request: It verifies $req.\sigma_{req}$ using $addr_{LC}$ associated with α by reconstructing the hash value from the request contents.

The origin of the response: It verifies $res.\sigma_{res}$ using $addr_{FN}$ associated with α by reconstructing the hash value from the response contents.

The match of the identifier: It verifies the identifier α in req matches the one in res.

The incorrectness of the response: To penalize the full node, the module must confirm that res contains incorrect information. This includes any of the following scenarios:

- Mismatch between the payment amount a in req and res.
- Outdated information due to return block height $res.m_B$ smaller than the one indicated by $req.h_B$.
- Invalid $res.\pi_\gamma$ when verified against h_{root}.

The verification logic of a fraud-proof in the Channels Management Module (CMM) is shown in Algorithm 2.

Algorithm 2 Fraud Proof Verification in the CMM

function FRAUDPROOFDETECTION
  Input: $req, res, addr_{WN}$
  Procedure:
    $reqDec \leftarrow$ decodeRequest(req)
    $resDec \leftarrow$ decodeResponse(res)
    // The match of the identifier
    require($reqDec.\alpha$ == $resDec.\alpha$)
    $chan \leftarrow$ getChannelInfo($reqDec.\alpha$)
    require($chan.T \neq$ "closed")
    // The origin of the request
    $h_{req} \leftarrow$ hash($reqDec$)
    require($h_{req} == reqDec.h_{req}$)
    require($chan.LC$ == recover(h_{req}, $reqDec.\sigma_{req}$))
    // The origin of the response
    $h_{res} \leftarrow$ hash($resDec$)
    require($chan.FN$ == recover(h_{res}, $resDec.\sigma_{res}$))
    // Payment Amount Check
  if $reqDec.a \neq resDec.a$ **then**
    slashAndReward($chan.FN$, $chan.LC$, $addr_{WN}$)
  end if
    // Timestamp Check
    $m_{req} \leftarrow$ getBlockHeightByHash($reqDec.h_B$)
  if $resDec.m_B < m_{req}$ **then**
    slashAndReward($chan.FN$, $chan.LC$, $addr_{WN}$)
  end if
    // Verify Merkle Proof
  $h_{root} \leftarrow$ getRootHash($resDec.m_B$)
  if verifyProof($h_{root}, resDec.\pi_\gamma$) $\neq true$ **then**
    slashAndReward($chan.FN$, $chan.LC$, $addr_{WN}$)
  end if
end function

VI. Implementation and Evaluation

We developed a PARP prototype for Ethereum with three components: a full node, a light client, and on-chain modules. The full node, built on Geth (version 1.13.12 [22]), adds 1827 lines of Go (v1.20) for PARP compatibility. The light client uses around 1500 lines of Go. 1631 lines of Solidity (v0.8.25) manage deposits, payment channels, and fraud-proof detection. The prototype supports full interaction between components. We release our implementation for open science [1].

The proof verification uses a Merkle proof. In our Ethereum-based implementation, Merkle Patricia Tries (MPT) generates a **state trie**, a **transaction trie**, and a **transaction receipt trie**, with root hashes stored in block headers. For the cost of submitting a fraud-proof, since Solidity cannot natively fetch root hashes for specific block numbers, the light client submits block header fields, including the trie roots, to regenerate the block hash on-chain. Ethereum's built-in block hash verification supports validation within the last 256 blocks, therefore, ensuring a trustworthy root hash.

We evaluated our prototype with these questions:

- Additional communication costs of a PARP RPC request and its response in terms of message size? (Sec. *VI-C*)
- Additional processing time caused by PARP, and how does it impact request processing latency? (Sec. *VI-D*)
- On-chain costs incurred by PARP? (*Sec. VI-E*)

[1] https://github.com/podiumdesu/parp-dev

- Additional CPU and memory are required by a PARP-compatible node compared to a standard Geth node as a function of the number of served clients? (*Sec. VI-F*)

A. Read and Write Workloads

A **read** workload includes requests that query and retrieve data from the blockchain without altering its state. It is typical for data verification and status checks.

A **write** workload refers to requests that change the state of the blockchain, typically by sending a signed transaction that will be validated and recorded on the blockchain.

In PARP, each request and response includes base layer RPC data with additional PARP metadata for accountability.

B. Test Setup

We established a local Ethereum network with three full nodes using Geth [22] as the execution client. One node ran a PARP-compatible Geth version processing the connection and requests from a light client. The network was deployed on local OpenStack virtual machines with 4 vCPUs and 8 GB of RAM each, including a similarly configured light client.

C. Communication Costs and Message Size

As shown in Figure 5, an RPC call is wrapped with PARP metadata to form a PARP request sent to the full node. The full node processes it, executes the call, and wraps the result with response metadata before returning it to the light client.

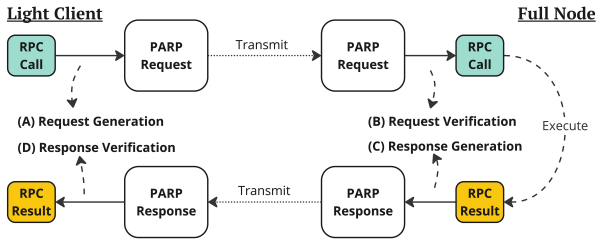

Fig. 5: Additional computation steps required in processing a PARP request and response.

For context, the size of an Ethereum JSON-RPC request for a raw transaction call, such as opening a payment channel, is **422** bytes, while retrieving an account balance is **118** bytes.

A PARP request includes two 65-byte signatures for transaction integrity (returned in the response) and payment authentication. The total overhead per request is 226 bytes.

A PARP response adds 187 bytes of metadata, including two signatures (one from the request), plus variable-sized proof verification data depending on the request type. The message size overheads are detailed in Table II.

	Size Overhead (in bytes)
PARP request	226 bytes
PARP response	187 bytes + Size of Merkle Proof

TABLE II: Message Size overhead for PARP RPC requests and responses compared to standard Ethereum RPC calls.

We evaluated the size of Merkle proofs for transaction inclusion within blocks. Write requests use the transaction trie root. As shown in Figure 6, Merkle proof sizes vary not only with **the number of transactions** included in one block but also with the **transaction index** within those blocks (explaining the sudden drop in the figure). For instance, for a transaction located in a block containing 200 transactions, the average Merkle proof size is approximately 1150 bytes.

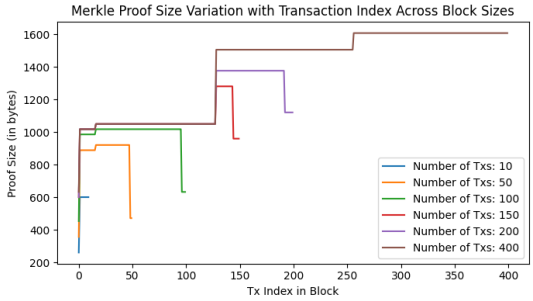

Fig. 6: Merkle proof size variation with transaction index across a range of different block sizes.

D. Computation Overhead and Latency

Our protocol introduces additional steps and computational overhead compared to standard Geth interactions (Figure 5). Table III details the additional processing time of the steps marked (A) through (D). The reported numbers are the average increase in latency for 100 requests for both a write and a read workload. For the write workload, we generated a transaction in a block with 200 transactions. For the read workload, we use an RPC request that retrieves an account balance.

Light Client Process Steps	**Write**	**Read**
(A) Request Generation	10.91ms	4.82ms
(D) Response Verification (proof)	7.13ms	5.78ms
(D) Response Verification (in total)	8.109ms	1.01ms
Full Node Process Steps	**Write**	**Read**
(B) Request Verification	714.43μs	703.13μs
(C) Response Generation (proof)	3.08ms	477.12μs
(C) Response Generation (in total)	3.37ms	1.29ms

TABLE III: Additional computational latency introduced by the protocol (average over 100 PARP RPC requests).

Action	**Gas Cost**	**MainNet (USD)**	**Arbitrum (USD)**
Full Node Deposit			
Deposit funds	45238	2.171	0.018
Channel Management			
Open a channel	196183	9.417	0.078
Close a channel	110118	5.286	0.044
Confirm closure	87128	4.182	0.035
Fraud Proof Detection			
Submit a fraud proof	762508	36.6	0.305
Median Transaction Fee (9/12/2024)		*1.606*	*0.350*

TABLE IV: On-Chain Cost Analysis of PARP Protocol.

E. On-chain costs

The PARP protocol includes several on-chain actions that incur costs in terms of gas fees. Table IV summarizes the gas costs and their corresponding USD values on the Ethereum Mainnet and Arbitrum L2 network. At the time of calculation, we assumed an ETH price of $4000 USD and gas prices of 12 Gwei for Ethereum Mainnet and 0.1 Gwei for Arbitrum.

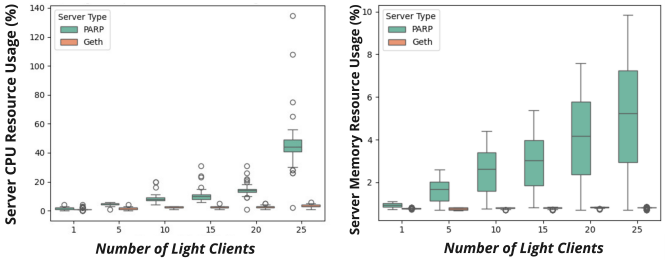

Fig. 7: Additional computation steps required in processing a PARP request and response.

F. Scalability and Performance Metrics

To evaluate scalability and performance, we tested a PARP-compatible full node with light clients sending two requests per second over two minutes. Using 4 vCPUs and 8GB RAM, the results in Figure 7, indicate that when the number of simultaneous light client connections reached 20, the average CPU usage was 14.3%, and the average memory usage was 2.63%. Even though these numbers represent a 3.43x increase in CPU usage and a 2.38x increase in memory usage compared to a standard Geth node, we argue that they remain within an acceptable performance range and support our claim that PARP can be integrated into real-world blockchain networks with reasonable overhead. We expect future optimizations could further reduce resource consumption.

G. Conclusion

Our evaluation shows that PARP introduces manageable overhead while ensuring accountability and reliability in the blockchain RPC layer. Communication costs remain reasonable, and processing time adds only minor latency for both read and write operations. On-chain costs are transparent and can be reduced with Layer-2 solutions like Arbitrum. PARP-compatible full nodes maintain acceptable CPU and memory usage, even with multiple light clients. Overall, PARP integrates efficiently into blockchain networks without significant performance loss, enhancing incentives and trust.

VII. Related Work

Several projects share PARP's goal of making the blockchain RPC layer more decentralized and permissionless. The Portal Network [23] employs the Kademlia DHT [24] to reduce the storage requirement for each node, allowing those with limited storage to contribute. This setup enables light clients to access specific network information on a peer-to-peer basis. However, the participation of nodes remains voluntary, with no mechanisms in place to reward serving nodes (in contrast to PARP's micro-payments approach). POKT [25] and Ankr [7] enable a network of decentralized nodes that serve requests in exchange for tokens but required to comply with Know Your Customer (KYC) regulations and to stake tokens. Ankr centralizes requests through a load balancer, while POKT requires users to access blockchain data through permissioned gateways. By contrast, PARP does not require the relay of requests through a centralized element and utilizes different incentives for node participation.

RPCh [26] introduces a decentralized gateway where permissioned entry and exit nodes relay calls to the HOPR MixNet [27]. It incorporates "Proof of Relay" to reward nodes for effective data relay and uses payment channels for cost-efficient rewards. While RPCh offers enhanced privacy via a MixNet, PARP avoids routing calls through a permissioned entry node and uses distinct cryptoeconomic incentives.

Olshansky *et. al* [28] have proposed Relay Mining, where RPC nodes charge fees based on request volume, with rewards drawn from tokens staked by dApps. Selected RPC nodes serve a particular dApp for a designated period, spanning several blocks, after which they submit their proof of work on-chain through a commit-and-reveal scheme verified by Sparse Merkle Trees. In contrast, PARP uses cumulative payment amounts in payment channels to calculate work performed, establishing a different incentive model.

Among all these initiatives, PARP stands out as the only RPC protocol that supports verifiable blockchain data access using a fraud-proof protocol, with micro-payments as direct compensation for serving requests.

VIII. Limitations and Future Work

Risks of single node dependence. In practice, a light client should connect multiple full nodes to enhance availability and avoid single points of failure. However, our protocol requires a light client to set up a payment channel individually with every full node it intends to connect with, adding costs and potentially discouraging multiple connections. Payment channel networks [29] could address this by avoiding opening a dedicated channel per client-server pair.

Privacy concerns in full node interactions. Another limitation of our protocol is its inability to fully address concerns related to full nodes snooping on sensitive request information, including content and network information such as IP addresses. Future extensions may employ cryptographic methods like homomorphic encryption [30] and commitments [31] for content privacy and Mixnets such as the Nym infrastructure [32] for anonymous communication.

Network rewards via Proof of Serving. PARP can form a new reward mechanism that we tentatively call "Proof of Serving', similar to Proof of Stake, but to incentivize full nodes to serve light clients. Payment proofs signed by light clients act as receipts, which full nodes can aggregate and submit to the network and claim a portion of the block reward. The main open issue is to address Sybil attacks whereby a full node controls fake light clients and connections. Introducing

a reputation system to validate the legitimacy of served light clients could be one solution to this issue.

Formalization of cryptoeconomic incentives. We have not yet addressed the details of the economic incentives enabled by the PARP protocol.

Two key areas for PARP's economic incentives include designing a fee schedule for RPC requests and linking full node stakes to the volume of client requests they can handle. Fee schedules must balance client affordability with fair node compensation, while staking thresholds enhance security against misbehavior. Formalizing economic incentives to enhance honest participation in blockchain systems has been the subject of many studies [33]–[35]. Most relevant to our work Moshrefi *et. al* [36] introduce fraud-proof mechanisms with slashing conditions that penalize data tampering. The cost model employed in "insured" cryptoeconomic security varies with the value of the transaction, aligning financial risk with transaction importance. The incentive formation principles detailed in this paper can enhance our protocol by integrating a formalized compatible incentive model.

IX. Conclusion

We have addressed the problem of the increasingly permissioned access to the serving layer of otherwise permissionless blockchain networks and its effects on the privacy, integrity, and availability of data access by application clients and end-users. Our **P**ermissionless **A**ccountable **R**PC **P**rotocol (PARP) extends the RPC protocol of blockchain full nodes to enable pseudonymous yet accountable interaction between light clients and full nodes. The protocol provides an alternative to permissioned and privacy-invasive but reputable NaaS providers on the one hand, and permissionless but less reliable public RPC nodes on the other hand. Our implementation and evaluation of the PARP protocol for the Ethereum network demonstrates the feasibility of a light client obtaining RPC service from a full node with high data security for the client, ensured payment for the full node, and acceptable computation and communication overhead for both parties.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback, which improved the clarity and quality of this work. We are also grateful to Glenn, Jan, and Kristof for their helpful comments and suggestions on the manuscript. This research was partially funded by the Research Fund KU Leuven and the Cybersecurity Research Program Flanders.

References

[1] Ethereum. (2024) Spin up your own ethereum node. [Online]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/run-a-node/#requirements

[2] ——. (2024) Nodes as a service. [Online]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/nodes-as-a-service/

[3] (2024) Web3 development platform — ipfs api &gateway — blockchain node service. [Online]. Available: https://www.infura.io/

[4] Alchemy. (2024) Alchemy. [Online]. Available: https://www.alchemy.com/

[5] M. M. (2022) My first impressions of web3. [Online]. Available: https://moxie.org/2022/01/07/web3-first-impressions.html

[6] M. Support. (2023) Why infura cannot serve certain areas. [Online]. Available: https://support.metamask.io/troubleshooting/why-infura-cannot-serve-certain-areas

[7] (2024) Ankr — the fastest web3 infrastructure. [Online]. Available: https://www.ankr.com/

[8] (2024) Quicknode - blockchain infrastructure powering secure, decentralized innovation. [Online]. Available: https://www.quicknode.com/

[9] (2024) Fast and reliable blockchain infrastructure provider - chainstack. [Online]. Available: https://chainstack.com/

[10] P. C., F. B., and K. C., "Sok: Blockchain light clients," in *Financial Cryptography and Data Security: 26th International Conference*, 2022, pp. 615–641.

[11] C. F. Torres, F. Willi, and S. Shinde. (2024) wallet-address-leakage-datasets. [Online]. Available: https://zenodo.org/record/8071006/files/wallet-address-leakage-datasets.zip

[12] ——, "Is your wallet snitching on you? an analysis on the privacy implications of web3," in *USENIX Security'23*, 2023.

[13] Aard. (2024) Awesome list rpc nodes providers. [Online]. Available: https://github.com/arddluma/awesome-list-rpc-nodes-providers

[14] (2024) Chainlist. [Online]. Available: https://chainlist.org/

[15] M. Support. (2023) What is infura, and why does metamask use it? [Online]. Available: https://support.metamask.io/networks-and-sidechains/

[16] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[17] B. Bünz, L. Kiffer, L. Luu, and M. Zamani, "Flyclient: Super-light clients for cryptocurrencies," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 928–946.

[18] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro, "Coda: Decentralized cryptocurrency at scale," *Cryptology ePrint Archive*, 2020.

[19] M. Hearn. (2013) Micro-payment channels implementation now in bitcoinj. [Online]. Available: https://bitcointalk.org/index.php?topic=244656.0;all

[20] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *FC*. Springer, 2020.

[21] V. Buterin. (2015) Ethereum foundation blog - state tree pruning. [Online]. Available: https://blog.ethereum.org/2015/06/26/state-tree-pruning

[22] Ethereum. (2024) Go implementation of the ethereum protocol. [Online]. Available: https://github.com/ethereum/go-ethereum

[23] ethereum.org. (2024) The portal network. [Online]. Available: https://www.portal.network/#/

[24] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*. Springer, 2002.

[25] (2024) Decentralised rpc — web3 infrastructure — pokt network. [Online]. Available: https://www.pokt.network/

[26] (2024) Rpch — private rpc provider. [Online]. Available: https://rpch.net/

[27] (2024) Hopr — blockchain data protection and privacy. [Online]. Available: https://hoprnet.org/protocol

[28] D. Olshansky and R. R. Colmeiro, "Relay mining: Incentivizing full non-validating nodes servicing all rpc types," 2024.

[29] N. Papadis and L. Tassiulas, "Blockchain-based payment channel networks: Challenges and recent advances," *IEEE Access*, vol. 8, pp. 227 596–227 609, 2020.

[30] R. Solomon, R. Weber, and G. Almashaqbeh, "smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption," in *2023 EuroS&P*, jul 2023. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/EuroSP57164.2023.00027

[31] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," 2019. [Online]. Available: https://eprint.iacr.org/2019/191

[32] Nymtech. (2024) The next generation of privacy infrastructure. [Online]. Available: https://nymtech.net/nym_litepaper.pdf

[33] E. N. Tas and D. Boneh, "Cryptoeconomic security for data availability committees," in *FC'24*, 2024.

[34] S. Deb, R. Raynor, and S. Kannan, "Stakesure: Proof of stake mechanisms with strong cryptoeconomic safety," 2024.

[35] A. Mamageishvili and E. W. Felten, "Incentive schemes for rollup validators," 2023.

[36] N. Moshrefi, P. Sheng, S. Deb, S. Kannan, and P. Viswanath, "Unconditionally safe light client," 2024.