Lava Network: Accessing blockchains

Yair Cleper, Gil Binder, Omer Mishael, Ethan Luc

April 18, 2024

Abstract

We propose Lava Network, a modular network for bootstrapping, scaling and optimizing access to any blockchain. Lava comprises an off-chain, peer-to-peer protocol and a Cosmos SDK appchain customized for blockchain RPC and APIs. The network works as a multi-sided marketplace, coordinating champions, chains, data providers, delegators and data consumers. Champions permissionlessly add support for new APIs and chains as "specifications', JSON files detailing the methods and computational cost to serve an API. Chains and sponsors deposit tokens to Lava Incentivized RPC pools, which incentivize data providers to join and offer performant service. Providers stake and add their nodes to the network, extending cryptoeconomic guarantees to the quality of their service and the integrity of their responses. Providers are also rewarded for high performance and accuracy. Lava aggregates node runners, their APIs and supported interfaces, serving as a smart router for RPC calls and sending requests to the best available node. Consumers such as dapps, wallets and indexers access Lava for data and can increase their usage limits by buying subscriptions in LAVA, the native asset of the network. With Lava's open source Server Kit, anyone can create a gateway and prioritize decentralization of the network. Finally, Delegators can stake and restake, increasing network security and the frequency of consumer pairings a quality provider can earn rewards on.

Lava supports a world of a thousand rollups, making it easy for chains to bootstrap node infrastructure and offer reliable and decentralized access to their ecosystem.

1 Introduction

1.1 Accessing a world of a thousand blockchains

Every blockchain and rollup must offer reliable access to its users and developers.

Access is the ability to connect and interact with a blockchain. This is increasingly important, with the modular stack now making it much easier to launch new blockchains and rollups. Each of these networks will require robust, fast and permissionless data infrastructure to onboard and scale an ecosystem of users and developers. To date, research into blockchain modularity has centered around data availability, execution, settlement and consensus. Access is critical, but overlooked.

Data infrastructure refers to the set of tools, APIs and protocols which facilitate blockchain access, with the most essential being a communication protocol called Remote Procedure Call, or RPC for short. RPC is how apps and users communicate with blockchains, fetching data and sending transactions by making calls to RPC nodes for that specific chain. For blockchain and rollup developers, it is essential that there is some set of node runners willing to expose RPC endpoints and serve requests for basic data.

With blockchains growing in popularity, the challenge for blockchain developers to scalably onboard dapp developers and users remains unsolved. Ecosystems have found new ways to fill this gap, working with major node providers or coordinating volunteer node runners to offer free service called "Public RPC". However, these solutions either lead to centralized control over the flow of traffic - see Ethereum - or a fragmented public RPC setup which is under-maintained and unreliable. And while it is true that "anyone can run a node" to avoid using public RPC, the system makes this both impractical and undesirable for all except the most technical of participants. As Moxie wrote in his famous takedown of crypto, "People don't want to run their own servers, and never will."[1]

This systemic issue has resulted in a broken and inefficient infrastructural landscape that leaves some ecosystems heavily permissioned by a small handful of providers, and other ecosystems unable to reliably serve demand. New rollups must bootstrap an RPC network from scratch or pay significant fees to providers. The majority of high profile Mainnet launches still see significant issues with RPC downtime. Providers are often hacked[2] and censorship issues abound on Ethereum, where node providers hold the power to block transactions and access to specific smart contracts.[3] Privacy exploitation cannot be underestimated; when the majority of traffic goes through only one or two providers, these vendors become the all-seeing eye, not dissimilar to the ads and data machines that have dominated the internet so far. Moreover, developers and users are often locked into their vendors, unable to easily switch providers during periods of downtime, despite the many alternatives that might be available.

As we hurtle faster towards a future and present of a thousand rollups, data infrastructure is now the most important frontier. It is clear that the current accessibility options designed to offer smooth onboarding to blockchains are severely limited.

The crypto ecosystem today faces the following inefficiencies:

1. **New chains and rollups struggle to bootstrap their data infrastructure** - New or smaller chains either spend huge amounts working with major providers or depend on small-scale, community node runners without clear metrics to judge and reward performance.
2. **Ethereum is centralized** - Established chains like Ethereum route the majority of traffic through a small handful of providers, vulnerable to attacks and acting as trusted third parties. Without automatic and simple ways to route to alternative providers, developers and users default to using one or two, creating a systemic censorship and access risk on major chains.
3. **Building multi-chain applications is fragmented** - dapp developers must work with and integrate a new provider for each chain they build on, taking time and resources away from their core product.
4. **Quality of service is opaque and highly variable** - it is permissionless to spin up a node for public service but performance and accuracy are neither enforced nor disclosed. Dapps require dynamic routing depending on the nature of user requests e.g. it is optimal to pick a provider closer to the location of the user request. Today, there is no accurate or automatic way to choose the best provider.
5. **Node providers are a single point of failure for dapps** - RPC calls are by nature sent to individual nodes with individual endpoints. These nodes can be hacked, act maliciously, or face downtime, leading to prolonged periods of poor user experience for newer chains with less experienced providers. Due to limited engineering resources, most dapps use only one node or one provider.

Lava is a modular network that any blockchain or rollup developer can use to permissionlessly bootstrap access to their chain. It introduces several novel concepts, including peer-to-peer, QoS-based request routing, optimistic conflict prevention and Inter-Blockchain Communication (IBC) protocol token pools [4], that make it easy for blockchains to offer reliable and performant RPC to their ecosystems.

2 Lava Network

The Lava Blockchain is the basis of the Lava Network. Lava Blockchain is a Delegated Proof-of-Stake (DPoS)[5] and fast finality IBC compatible appchain built using the Cosmos SDK and CometBFT, a Tendermint-based Byzantine fault-tolerant consensus algorithm. Lava Blockchain acts as a settlement layer for off-chain remote procedure calls (RPCs)[6] between consumers and providers of data. Consumers connect to providers peer-to-peer with both finalization proofs and rewards claims submitted on-chain. Many basic functions of the Lava Blockchain operate similarly to other Cosmos / CometBFT blockchains and use common modules. Similarities notwithstanding, there are many features which are unique to Lava Blockchain and documented below.

2.1 Overview

Lava Network[1] is a modular network serving as the access layer for all blockchains. Remote Procedure Call (RPC) data is the first supported use-case, but broader implementations of data access are possible and already available, such as subgraphs and indexing. The protocol is composed of specifications – governance-defined modular objects that specify the collections of APIs which data providers support on the network. Specifications can be modified, updated, replaced, or added through acts of on-chain governance.

Broadly, Lava Network functions in three distinct ways:

Coordination engine – Blockchain and rollup developers add the specifications for their chain to Lava, and data providers will join the network to serve RPC. This creates a dynamic and multi-chain infrastructure layer which can quickly coordinate providers for any existing and newly launched chain. As demand increases, the supply of providers will also increase.

Marketplace – Data providers across all chains compete to provide a broad range of services. Lava is a permissionless network which allows the dynamic inclusion and support of any API and interface. Services such as Ethereum RPC or token APIs are defined within modules called "specifications", which can support add-ons and extensions such as archival requests. This approach gives consumers maximum flexibility and choice in one place, allowing the construction of custom subscriptions that suit the needs of any individual consumer.

Smart Router – Data providers across all chains also compete to provide the most performant service. Consumers are directed to optimal providers based upon provider quality of service and the priorities of a given consumer. Consumers can define strategies which prioritize specific provider attributes such as response latency, accuracy, availability or overall strategies such as distributing requests across providers for privacy. Consistency is enforced, while conflicts are detected off-chain and resolved on-chain. Data access efficiency is optimized by choosing intelligently among providers on the network.

2.1.1 Solution

Lava is designed to solve the five issues described in the **Introduction** (Subsection 1.1) :

1. **New or smaller chains have difficulties bootstrapping RPC node runners** - chains and sponsors can deposit token incentivizes, attracting a network of providers or leveraging Lava's existing network. These providers will compete to meet user demand.
2. **Established chains like Ethereum route the majority of traffic through a small handful of providers** - Lava serves as a decentralized network of node providers for every chain.
3. **Fragmented multi-chain infrastructure** - the Lava Protocol abstracts away all interfaces and chains, and developers' multi-chain calls are routed to relevant providers on the network
4. **Quality of service is opaque and highly variable** - reputation scores of providers on Lava are on-chain and impact future pairing frequency. This crypto-economic approach creates a standard of performance across the network. Variance is further mitigated by efficient routing of requests.
5. **Node providers are a single point of failure for dapps** - Lava provides dapps with a pairing list of available providers, and relays are entirely peer-to-peer. Even if the Lava Blockchain halts, dapps can still get responses from providers. If there is a provider for the required chain on Lava, developers can send relays.

[1] A detailed graphic of Lava Network's blockchain and protocol appears in the **Appendix** (section A) (figure 5)

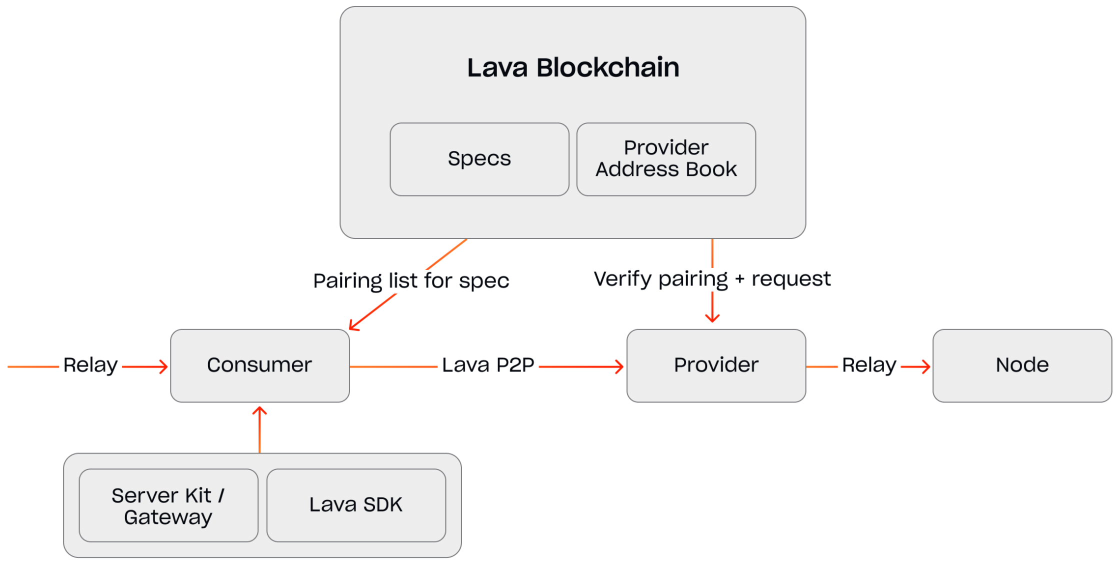

Figure 1: A simplified overview of the protocol

2.1.2 Data Access

Data access is conducted through CLI (local client), web UI (gateway interface), or programmatically (software development kit). The implementation details of each method of connecting to the protocol to exchange data vary, but the underlying mechanisms of the Protocol are consistent. Lava Protocol provides data access through specifications. Specifications are largely unopinionated and can be configured to conform to various interfaces and styles of data transfer.

Lava Protocol connects consumers to providers using PairingLists(PL). Providers on a PairingList are determined algorithmically based upon several factors. Primary factors are provider stake, geolocation, latency, availability, and freshness (sync). Secondary factors may involve a consumer's selection strategy, or communicated preference for or against specific providers. A closer look at this process is available in the pairing section of this paper – but a simplified view is available here:

- First, consumers receive a pairing list composed of the top providers, as ranked by stake and quality of service, lasting for multiple blocks (an epoch).
- Second, consumers establish a peer-to-peer connection with the most optimal provider from the pairing list.
- Third, consumers send and receive data.
- Fourth, upon epoch completion, providers accumulate compute units (CU). At the end of the month, they claim their share of the distributed rewards of a subscription relative to its cost.

2.2 Key Roles

2.2.1 Data Providers

Providers stake and earn LAVA for serving data on the Lava Network. Providers use the provider kit to serve relays across different interfaces on Lava Protocol according to a given specification. A provider, in many, but not all, cases, will also be a node runner for a specific blockchain.

2.2.2 Data Consumers

Consumers spend LAVA to consume data on the Lava Network. A consumer is anyone who consumes web3 APIs using Lava Protocol. Examples include developers, wallets, dApps, exchanges, indexers, and more. Consumers use the LavaSDK, ipRPC endpoints, Gateway Endpoints, or Server Kit to retrieve data.

2.2.3 Validators

Validators earn LAVA for securing the chain. Validators function similarly to other Cosmos chains: proposing blocks, voting on blocks, and validating state.[7]

2.2.4 Champions

Champions earn LAVA for creating, maintaining, servicing, and supporting specifications on the Lava Network. Many champions create specifications and propose them on-chain, maintain existing specifications in light of necessary updates or changes, or write software that serves specifications (node clients, API clients/ indexers, or other innovations). A portion of rewards is reserved for Champions.

2.2.5 Delegators

Delegators stake LAVA to providers and validators on the network. This makes the network more secure and allows delegators to partake in providers' and validators' risk in exchange for rewards.

2.3 Key Concepts

2.3.1 Incentivized Public RPC

Chains and sponsors can create Incentivized RPC pools on Lava, depositing any token to attract, scale and incentivize access to high quality node providers. The result is a public good offering free and reliable access to the corresponding blockchain of the Pool.

Data providers join Lava and are paid out in any selected token used to fund the pool. Data consumers such as developers receive an endpoint which uses Lava to route between different providers. The flow for blockchains to onboard to Lava is as follows:

1. Blockchains and sponsors create an Incentivized RPC Pool on Lava, typically depositing their native token
2. Data providers join Lava and become available to data consumers
3. The Incentivized RPC Pool attracts, scales and optimizes providers by rewarding performant service
4. Existing node runners can also join to monetize their current infrastructure
5. Quality of service metrics and service records are recorded on chain
6. Rate-limited, redundant RPC access to developers, dapps, and data consumers with no associated usage cost is offered
7. Incentives are distributed on-chain to all participating providers, according to the amount of data served, eliminating vendor lock-in

dApps and Developers access the Lava network to connect peer-to-peer with the best providers on Incentivized Public RPC interfaces for free.

2.3.2 Specifications (Specs)

Specifications (specs) are modular blueprints for Lava's multi-chain and multi-API support, defining chain and method requirements, costs, verifications, and numerous other configurations. Specifications define the requirements that providers must meet to serve a specific API's data. Specifications may also define optional behaviors for providers including extensions (modified default behavior) and addons (opt-in supplementary behavior).

Extensions modify default behavior of existing API collections. An example use-case is archival nodes which have the same APIs as other nodes, but different expected output. For example, an Ethereum archive node will have an existing interface for `eth_getBalance`, but its possible results will go back further. As a result, providers who serve this specification and meet that requirement are more highly rewarded. The specification for the extension accounts for these differences from normal behavior.

AddOns add new behavior and API collections. An example use-case is debug nodes which have additional API methods that a standard node would not support. For example, an Ethereum debug node needs to support interfaces for `debug_traceBlock`, `debug_traceBlockByHash`, in addition to the standard interfaces of an ordinary node. Serving these APIs is an additional computational expense to providers, while offering consumers a world of new functionality. The specification lists these added functions as a supplementary API collection.

Specifications define not only chains but also provider requirements and contain verifications that ensure providers deliver the services promised by the protocol. Network participants can query specifications directly on-chain to understand supported APIs. The tree structure for a specification (JSON) contains numerous fields, some of which are highlighted in Table 1 below:

Title	Description	Examples
`Index`	a unique identifier for a specification, used as universal reference	LAV1; AVAX; SOLANA
`Name`	the long-form human readable title for the specification	lava testnet; avalanche mainnet; solana mainnet
`Imports`	the indices of specifications, for which API Collections will be inherited	ETH1, COSMOSSDKFULL
`Contributor`	a list of Lava addresses[8] of those who create, update, and maintain the specification	lava@1jqfvjtkrvz08z4g6adceru3fqv8z29859lkdss, lava@1q3nrxtn2zljgj5f9mpeu06qz838wzsfk0u6l3f
`ContributorPercentage`	a percentage of yield that is allotted to addresses listed in contributors from use of the specification	2%
`MinStakeProvider`	the minimum stake in LAVA tokens a provider must make to qualify for a pairinglist relevant to this specification	{“denom”:“ulava”, “amount”: “47500000000”}
`AverageBlockTime`	the typical duration, in *ms*, for a new block to be created on chain	2500
`BlocksInFinalizationProof`	the number of blocks in a finality proof	3
`AllowedBlockLagForQoSSync`	the threshold (in number of blocks n) for which a provider can have latest block $>= n$, before being considered out of sync with the chain	4
`APICollections`	the mandatory APIs, as well as the addons, extensions, verifications, and parse directives to be supported by a provider	collection of ETH APIs

Table 1: Specification Descriptions

2.3.3 Compute Units (CUs)

Compute Units (CUs) are a numerical representation of the computational difficulty of executing a specific API call. Every API executed on the protocol has a nominal cost in compute units. Compute units are useful for several calculations performed by the protocol: to calculate provider rewards, set the default timeout sensitivity on a call, establish bonuses for rarely supported APIs, and to throttle consumer usage according to subscription. Compute Units are also broadly used as a proxy for the amount of work performed on the network.

2.3.4 Lava Token (LAVA)

The **Lava Token (LAVA)** is the native asset of the Lava Network. It has multiple purposes: 1) it is the reward for Providers, 2) it is the consumers’ means for acquiring subscriptions and paying for compute unit usage, 3) it is the reward for validators who secure the blockchain, 4) it is the reward for Champions who create and maintain specifications, 5) it is used to pay for gas fees, 6) it is used as a governance token allowing token holders to participate in network decisions, 7) it is used to stake providers and validators - offering resistance against sybil attacks. The set of incentives which connects providers and consumers depends upon the Lava token. Lava token is compatible with Cosmos wallets. $\mu Lava$ (or uLAVA) is frequently used as a lesser unit (0.000001) of the same denomination and appears throughout this paper.

2.3.5 Epochs and Sessions

Epochs are batches of consecutive blocks. Blocks are grouped together to establish periods of pairing – whereby an epoch can be defined as the number of blocks n for which a Consumer will use a pairing list. Thus, PairingLists are fixed within each epoch.

Sessions are continuous exchanges of relays between a consumer and a provider. Consumers often maintain multiple concurrent sessions with various providers and vice versa. Sessions are a consequence of pairing and exist primarily within the confines of a given epoch.

2.3.6 Pairing Lists (PL)

A **‘PairingList’ (PL)** is a list of providers for whom a consumer can send and receive data. Within an epoch, a given consumer accesses a unique PairingList - representing a subset of the providers available on the network for services. Providers are chosen for PairingLists by many factors including geolocation, reputation, and providers’ total stake. The construction of Pairing Lists is covered in more detail later (subsection 4.3.2).

2.3.7 Relays

Relays are the protocol’s fundamental service delivery operation; all data exchanged between providers and consumers is carried within relays. Relays consist of both data and metadata. Relay metadata is used to carry information about the quality of the service provided and its veracity, while relay data conveys the payload of a relevant request. Relays are not conducted on-chain, but use the blockchain for pairing as outlined in earlier sections.. Protocol buffers (protobufs) are used to define message structure for relays and messages are sent over gRPC[9]. Consecutive relays are grouped within a single session.

2.3.8 Restaking

Restaking is an innovative mechanism introduced by the `x/dualstaking` module to enhance the reward system and governance on the network. Restaking provides an avenue for delegators to earn additional yields by extending their stake to both providers and delegators, thus amplifying their benefits. Restaking is covered in more detail later (subsection 7.6).

3 Access and Connectivity

3.1 Onboarding Consumers

There are multiple means for consumers to access the Lava Network.

3.1.1 Software Development Kit (SDK)

Lava SDK provides developers programmatic access to Lava Network using TypeScript interfaces. Developers provide either a private key or a badge (subsection 3.1.2) for the SDK to initialize. The SDK validates the users' credentials upon initialization and then executes relays without requiring a user to input URLs. A developer using the SDK can freely send relays to any of Lava's supported APIs. Decentralized applications (dApps) built using LavaSDK rely on the network to supply the most appropriate RPC provider automatically, rather than supplying endpoints manually. Common web3 software development kits such as web3.js, CosmJS, Ethers.js, or viem have integrations with Lava which displace the need for hunting down RPC URLs. LavaSDK works by getting a pairing list using Lava nodes running on the Lava network – then separately executes RPC relays.

3.1.2 Badge Server

The Badge Server generates temporary credentials (i.e. badges) which can consume CU on Lava network. A badge allows any consumer to make calls to any supported API without using or providing their private key. Any consumer can run their own badge server. Importantly, badges are useful for front-end development, since they provide a straightforward way for web applications to harness the Lava Network directly from the browser.

3.1.3 Gateways

A Gateway provides a hosted point of access for the Lava network. Gateways may run Server Kit (subsection 3.1.4), or any compatible protocol interface, as a backend and serve client requests in their original form. In web3, most developers are accustomed to direct access to RPC using URL endpoints. Gateway users own their account on chain and can replace the gateway at will. Gateways are *sufficiently decentralized* following the paradigm set by Farcaster [10] Gateways offer a familiar developer experience for the sake of convenience. Anyone can propose a Gateway provider on Lava through a governance proposal. A diagram is shown below:

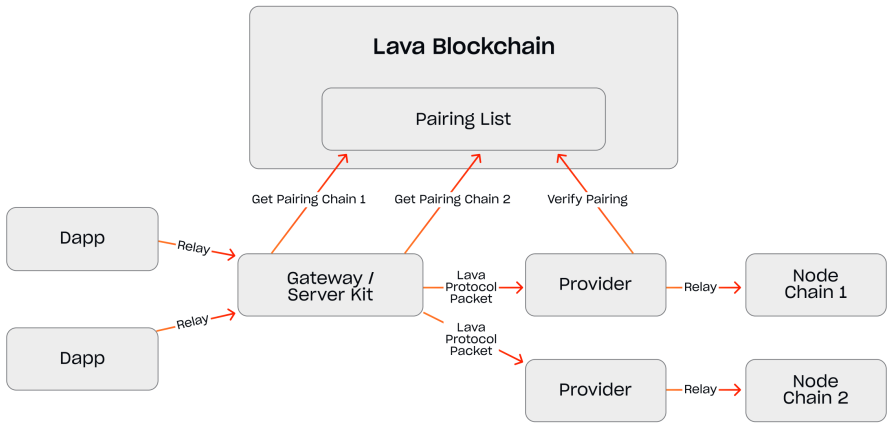

Figure 2: Dapps interface with a Gateway to access providers on the Network

3.1.4 Server Kit (aka RPCConsumer)

The Server Kit provides a hosted RPC interface for consuming data from the Lava Network. Written in Golang and highly concurrent, the Server Kit allows for the connection of multiple high throughput front/backends to be connected to the Lava Protocol simultaneously. It is ordinarily used as the backend for Gateways and strives to have full parity with LavaSDK. The ServerKit can be useful in the construction of consumer-facing web products where there is a need to partially abstract away interaction with the Lava Protocol - as for when one wants to construct a web application which services RPC requests from end-users who do not need to interact directly with the protocol. The Server Kit is tailor-made for these many-to-many RPC relationships.

3.2 Onboarding Providers

Providers onboard to the network through a series of steps.

- First, providers identify a specification that they would like to provide service through. For example, a provider might want to serve JSONRPC data over Ethereum Mainnet (`ETH1`), or serve archival data on NEAR Testnet (`NEART`). If the chain or API is not currently supported on Lava, they can become champions and create a specification for providers to use.
- Second, a provider runs the provider process and configures it to point to their node. Their node must conform to the requirements of the specification. For more details, see Provider Conformance.
- Third, the provider stakes on-chain on the specification, exposing the endpoint of its provider. Once staked, a provider is visible to the protocol and will be quickly and seamlessly selected by consumers after inclusion in a subsequent epoch's pairing lists.

3.3 Adding Blockchains and APIs

Lava is limitless in its extensibility and support. Specifications (subsection 2.3.2) are both modular and composable, which makes adding new chains and APIs to the protocol easy. Developers and technical contributors describe specifications in JSON proposals that are submitted for approval. Past specifications can be used as scaffolding for new specifications (referred to as `imports`). Technical guides and resources are made available so that the process can be completed permissionlessly. Specifications are proposed through governance in an on-chain transaction:

lavad tx gov submit-legacy-proposal spec-add [specjson1], [specjson2] –from [user] [gas-flags]

Once accepted, anyone who fits the parameters can join the network as a provider of the described service and serve APIs as defined in the specification. Providing RPC is a common use case of specifications, but it is not the only one. Any data exchange which conforms to Lava's structure can work. This is quite unlimited and includes abundant potential future use-cases: indexing solutions for rollups, large language models and artificial intelligence networks, intent solvers, cross-chain decentralized transaction bundling, light client incentivization, decentralized sequencers and oracles, secure wallets and nodes, and seed node accountability. Each of these is explored in detail in the section preceding the conclusion (section 8).

4 Discovery and Pairing

4.1 Consumer Conformance

Lava protocol uses subscriptions to allot consumer allowances of compute resources. Establishing a subscription is mandatory for consumers who seek to pair with providers on the network. Subscriptions are inspired by traditional SaaS packages and are an alternative to the pay-as-you-go model; they ensure predictable revenue for providers and stable, foreseeable service delivery for consumers, at regular intervals with no chance of consumption overages. Each data consumer needs only a single subscription and can get access to every supported chain by the Lava Network.

Fine-grained control over consumer usage is possible with subscriptions. A detailed description of each part of subscriptions is explained below:

- **Subscriptions** are time-demarcated packages, which enable a consumer to pair with providers and perform data exchanges on the network. Subscriptions operate on a monthly basis, starting from the block after purchase. A subscription works by enacting a monthly allowance for compute unit expenditure. If, at the end of a month, a consumer has more months remaining, the monthly allowance is reset to the selected subscription plan's monthly allocation.
- **Projects** organize and control resources within a subscription. Once a subscription plan has been purchased, a consumer can create projects under that plan. Whenever a consumer buys a subscription plan, an admin project is autogenerated with the default index, `<subscription_owner_address>-admin`. Projects track the used compute units, and contain a list of project keys that are subscribed to a plan.
- **Project Keys** are the list of accounts that can use or alter a given project. Each project key consists of two components: a valid Lava address[8] and a key kind. Project key kinds are either 1 for `ADMIN`, 2 for `DEVELOPER`, or 3 for `ADMIN+DEVELOPER`.
- **Admin Keys** are project keys which can edit the project's admin policy, enable or disable the project, and add or delete other project keys. An admin key corresponding to the subscription creator is added to a project by default. Admin keys are ordinarily for management purposes and do not allow for consumption of allotted compute units. There is a combined `ADMIN+DEVELOPER` type specified that allows admins to consume CUs on a project.
- **Developer Keys** are project keys that allow a consumer to use the allotted compute units of a project. Consumers can only use their public address in the developer key of one project. The intended usage is for dapps and services which consume APIs to have unique identifiers on the network's back end. If a project needs to provision services to multiple users, badges are the recommended approach.
- **Badges** are ephemeral credentials that grant limited access to compute units. A developer key can be used to generate a badge, which must be signed by an external badge-server. The intended usage is for dapps and services which consume RPC to have unique identifiers for end-users. A badge is only valid for a single epoch, must be signed by a developer key to establish proof of grant, and is allocated a compute unit limit.
- **Policies** are a set of limitations that affect the terms of a subscription. Plans always have policies, which are established by default. Policies can be used to limit total consumable CUs, limit consumable CUs per epoch, and set the maximum number of providers with which to pair.

Consumers can purchase and configure their subscriptions to accommodate patterns of usage, and consume APIs from available providers on the network.

4.2 Provider Conformance

Any provider joining the network and attempting to be listed on a pairing list must conform to a valid specification. Lava's provider client will not allow a provider on the network which fails verifications and/or does not support mandatory interfaces on the relevant specification. Consumers can also report non-conforming providers to get them taken off-line (jailed) or even lose rewards (slashed) in the future. Specifications are thus the minimum barrier for entry that a provider can pass to deliver service on the lava network.

Additionally, providers can view specifications to get a better understanding of rewards and service expectations for a given API. A separate provider stake action is required for each supported specification:

lavap tx pairing stake-provider [index] [amount] [provider-address] [provider-domain-address] [geolocation] [validator-lava-address]

The provider staked amount must exceed the `MinStakeProvider` (subsection 2.3.2) (table 1) in order to successfully stake on a given specification. This does not mean the provider must exceed the threshold by themselves; if total delegation (i.e. a provider's effective stake) exceeds the `MinStakeProvider`, the provider can serve relays regardless of the amount they staked with the `stake-provider` command.

Broadly, specifications are key to any provider understanding service and staking requirements. In regards to providers, specifications define:

- Every **mandatory API collectio**n a provider must support
- Every **optional API collection**, either as AddOns or as Extensions
- All **verifications** a provider must pass to ensure working endpoints
- The **minimum stake** required to offer services on Chain
- **Finalization criteria** for provider finalization proofs
- Details of **enabled API calls** such as their CU cost, bonus multipliers, and whether they are deterministic

4.3 Pairing Methodology

Lava protocol employs a time-based mechanism known as "pairing," with a predetermined time interval, currently set to 15 minutes and called an epoch, determined by the network's governance. Pairing is the primary method of discovery for consumers looking to engage providers' services on the network. Pairing results not just in the generation of a list, as defined in Key Concepts, but the correct matching of providers with consumers on the network. To accomplish pairing, consumers must possess a valid subscription or project and providers must match a specification and be staked on-chain. Communicating with pairing is handled in two-parts: the generation of pairing lists via matching algorithms in use by the PairingEngine, and the process of provider selection via selection strategies of the ProviderOptimizer (subsection 6.5).

4.3.1 Primary Factors

There are several factors which directly affect the likelihood that a provider is selected for a PairingList (PL):

Consumer Preference - providers are filtered by a consumers' subscription policy preferences. (subsection 4.1)

Provider & Consumer Geolocation - providers are selected based upon their identified geolocation matching a consumers' identified geolocation.

Provider Stake - providers' staked amount increases chances of selection for a given PL.

Provider Reputation - providers' performance with other consumers (subsection 5.3.2)

Consumer Profile - type of consumer - as identified by Clustering (subsection 5.3.3)

4.3.2 Generating Pairing Scores

Consumers connect to providers by accessing generated and verifiable lists of available providers known as a 'PairingList' (PL) (subsection 2.3.6). Pairing lists are dynamic, deterministic, and pseudorandom [11][12]. The process for acquiring one is sophisticated:

1. The consumer client sends a GetPairing query to a Lava Node. Alternatively, the consumer client submits a request to a static provider for a pairing list or uses a Lava provider on the Lava network to acquire a pairing list from the protocol.
2. The pairing engine uses the parameters of the consumer client to perform informed matchmaking and generate a list of providers eligible for pairing.
3. Filters are applied. Unwanted providers such as those who are frozen/jailed, do not support required Add-on/Extension APIs, or of distant geolocation are filtered out. Providers can also be filtered out by an applicable plan or consumer project policy which has a set list of 'SelectedProviders.'
4. The remaining, filtered providers are pseudorandomized using a changing seed, unique per consumer and derived from Lava blockchain hashes, as a salt.
5. Providers are assigned scores based on Quality of Service metrics (latency, availability, freshness/synchronization) and weighted by provider stake.
6. A Pairing List that is valid for the duration of one epoch with the highest weighted scores is returned to the consumer client.

Step 5 explains that Providers are assigned scores, but does not explain how. The PairingEngine combines provider's stake information, reputation and geolocation. The scoring process goes as follows:

1. Pairing Requirements are inherited from applicable consumer policy.
2. The consumer client generates pairing slots with requirements, where one slot is reserved for one provider.
3. The pairing score of each provider is computed with respect to each slot.

The provider returns a list of scores applicable to every consumer sharing the same set of filtering policies, the same geolocation, and stake.

4.3.3 *PL* Slot Selection

The *PL* Slot Selection process is designed to allocate providers to consumer slots in a fair and balanced manner, using a pseudorandom number generator to ensure diversity and mitigate potential censorship or spear-phishing attempts. The process is deterministic, ensuring that a given *PairingList* (PL) can be recalculated using specific parameters.

First, each slot on a given *PL* is filled with a pseudorandom number generator; it gets a seed based upon the following: 1) the current epoch, 2) the projectid of the consumer in question, 3) the hash of the first block of the epoch.

Once a provider is selected for a slot, they cannot be selected for the same pairing list again. Once all slotss are filled, a PairingList is returned to the consumer client.

A given pairinglist (*PL*) can be recalculated using a consumer's address, the specific block it was requested at, and the specification `Index` of the blockchain data was requested from. *PL*s are regenerated per epoch and prerequisite to provider selection.

As opposed to ***PL* Slot Selection**, *Provider Selection* is used to refer to deciding which provider will receive which relay. Provider selection is controlled by the Provider Optimizer and handled separately from assigning PairingList slots. The ProviderOptimizer uses a configurable selection strategy, discussed later, to select the most optimal provider. For more information see **Selection** (subsection 6.5).

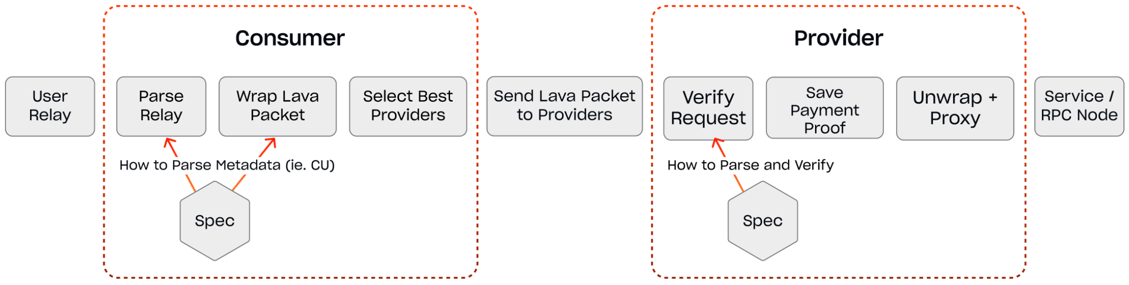

Figure 3: Consumers and Providers engage in discovery and pairing in order to exchange data on the network

5 Service Delivery

In every data protocol, one of the most important pieces is the accuracy and reliability of the data. Lava Protocol provides mechanisms for peer-to-peer service delivery. Once data has been successfully exchanged, the record of that exchange is recorded on-chain. The on-chain record is used to verify exchanges took place and settle payment for service providers.

5.1 Accountability and Verifications

A relay carries information for both verifying its validity and ensuring provider and consumer accountability. Each relay session will carry a quality of service report specific to relays conducted within that session, a unique session identifier, a spec identification, a content hash, a summation of all compute units expended within the session, and the number of relays conducted within that session. Additional information, such as a reputational quality of service report (called excellence), relevant party signatures, finalized block hashes, the number of the latest block, and a list of unresponsive providers is also provided in parallel. Cumulatively, this information is used by the protocol to assure that data exchange has been both accountable and verifiable. Beyond this, providers and consumers must both use private keys to exchange data on the network. The private key of a given provider and the private key of a given consumer are both used to sign relays exchanged between them. Data is signed by private keys on three different occasions: 1) Relay Session Data, 2) Relay Data Response, 3) Finalization Proof.

A key to establishing provider truthfulness is finalization proofs. Finalization Proofs accompany every relay sent on the network. Finalization proofs contain finalization data which is composed of the latest block a provider has seen and the provider's proof of the latest block. On fast finality chains, a hash of the latest block is sufficient for proof. However, on blockchains where finality may be in question, a provider must provide hashes of blocks even further back. The number of blocks needed for finalization is defined in the specification as `BlockDistanceForFinalizedData` and can be configured based on the likelihood of reorganization.

In addition to signatures, one or more section(s) of each specification outlines verifications that a provider must pass in order to provide services on the network. When a provider starts the provider process, the process retrieves the specification from the blockchain and evaluates the endpoints of the provider against the specification's mandatory APIs. Verifications, listed in the spec, are used, via parse directives, to elucidate what correctly parsed data should be returned for certain calls.

5.2 Communication Abstraction

Lava facilitates relay requests indiscriminate of the structure of the underlying data. Consequently, a wide variety of data profiles are be supported by the protocol.

5.2.1 Chains

Irrespective of idiosyncrasies of architecture, blockchains use and consume data by passing information between nodes to establish consensus. Remote Procedure Calls (RPC) are the standard by which most consumers access this data from nodes on various chains. Generally, the unique structure of calls is secondary to their function; all data is transferred over one of several standardized interfaces. Because of this fact, Lava protocol exhibits measures for chain abstraction. LavaSDK, Lava Gateway, and the Lava consumer client gain access to data similarly regardless of underlying chain architecture. Moreover, blockchains which share common architectural features are easily implemented on Lava by use of specification inheritance (called importing). As a result of these imports, with each added blockchain specification, new chains are easier to implement.

5.2.2 Application Programming Interfaces (API) Formats

Lava protocol supports a diversity of interfaces:

Interface	**Via URL**	**Via SDK**	**Websockets**
JSON-RPC	✓	✓	✓
REST	✓	✓	**x**
TendermintRPC	✓	✓	✓
URI	✓	**x**	**x**
gRPC	✓(HTTP/2 only)	**x**	**x**
Web-gRPC	✓	**x**	**x**

Table 2: Supported Interfaces

Ethereum Virtual Machine (EVM) chains, Cosmos (IBC) chains, and proprietary formats inevitably use one or more of the supported interfaces.

5.2.3 Smart Routing

Smart routing is the process of directing specialized consumer requests to specialized providers which can handle them. Filters are applied during Pairing that disqualify providers who cannot reply appropriately to a given consumer's request(s) (subsection 4.3.2). Consumers in search of extended functionality sometimes submit specialized requests over RPC (i.e. requests for archival data that require the unpruned history of a blockchain). Under a traditional RPC schema, this would involve locating or gaining access to a node which could handle the special requests and directing the relay requests which require special treatment to that node. This happens most often when a consumer decides to use expensive APIs. On Lava network, a consumer can direct specialized requests to the protocol and have them routed to the appropriate parties without concern. With smart routing, specialized providers such as archive nodes do not get requests that a regular node can handle. As a consequence, the network has better provider resource utilization. Specifications define modifiers and bonuses for providers who serve these special APIs, which further ensures that providers receive extra rewards for specialized service.

5.3 Quality of Service Scoring

Quality of Service (QoS) scoring is a system designed to assure quality service exists on the network. It is achieved through two separate but similar means: Passable QoS and QoS Excellence. Both of these measures are attentive to three criteria: latency, availability, and sync. After measuring and approximating scores across these three criteria, provider rewards and provider-consumer pairing are adjusted dynamically.

Criteria	**Definition**	**Metric**
Latency	the amount of time elapsed before a request is returned	ms
Sync	the provider proximity to the latest block on a specific chain	block distance, as derived from latest block number
Availability	the tendency of a provider to respond to requests received	% of answered requests in a session

Table 3: Table of Criteria for Scoring

5.3.1 Passable QoS

Passable QoS is an iterative algorithmic consumer-side scoring mechanism which provides per-session provider scores across all aforementioned criteria. Its goal is to ensure that providers on the network are up to an operable standard. Passable QoS is represented by a single coefficient which is computed by a geometric mean of latency, sync, and availability scores. The details of each of these computations is mathematically advanced and is accomplished via block interpolation, scaled averages, and numerical thresholds. The end result is a simple score between 0 and 1 which represents provider performance on a given session. As stated the Passable QoS score is the geometric mean of three sub-scores (latency, sync, availability). Each of the three areas is calculated algorithmically without direction of the consumer. They are taken together to make a Passable QoS Score that will be 0 for complete failure, 1 for perfect performance, or, most likely, some number between 0 and 1. The table below explains how each is calculated:

Each of these respective scores is an aggregate over a session and can be 0, 1, or any number between 0 and 1. Because the Passable QoS Score is calculated as the geometric mean of the previous sub-scores, a 0 of any sub-score will lead to an overall score of 0. This is reasonable to consider. A provider which has unreasonable latency, whose blocks are wildly out of sync, or who is unavailable to answer responses cannot possibly be considered as passable quality of service. Naturally, this antagonizes the worst performing providers on the network. A final Passable QoS Score is calculated like this:

$$PassableQoSScore = \sqrt[3]{AvailabilityScore \times LatencyScore \times SyncScore}$$

Criteria	Scoring	Metric
Latency Score	Latency of $> x$ ms will produce a score of 0 and latency $\leq x$ ms will produce a score of 1, where x is defined in the spec as the latency threshold for a given call.	Absolute Threshold (0 or 1) per relay - averaged over a session
Sync Score	If the provider's distance from the median latest block of all providers is greater than `AllowedBlockLagForQoSSync` in the spec, the provider receives a score of 0. If the provider's distance $\leq$ than the `AllowedBlockLagForQoSSync` in the spec, they receive a score of 1.	Absolute Threshold (0 or 1) per relay - averaged over a session
Availability Score	In a given session, a provider must respond to no less than 90% of requests. If a provider responds to 90% or less of requests, they receive a score of 0. However, if they pass the minimum threshold of 90%, they receive a score scaling from 0 at 90% all the way up to 1 for 100%.	Scaled Average (0 to 1) - biased linearly

Table 4: Table of Criteria for Passable QoS Scores per Relay

Further, Quality of Service is assured by attenuating provider rewards in response to these final scores; a maximum score of 1 reaps full rewards whereas a minimum score of 0 reaps half rewards (50%). The rewards on that session are adjusted by the protocol based upon the providers' performance:

Passable QoS Score	Reward (%)
0	0%
0.125	50%
0.25	56.25%
0.5	62.5%
0.625	62.5%
0.75	87.5%
1	100%

Table 5: Passable QoS Score and Reward

The score can be any number between 0 and 1, thus reward percentages are not restricted to those depicted in the chart above. Note that reward adjustment applies only to the session during which the score was calculated and has no effect on future sessions or pairing opportunities of the provider. The provider is guaranteed a partial payment (50%) upon returning any response so that there is always an expense to the consumer and that malicious actors cannot overcome computational expense through providing false reports. Passable QoS Scores are self-reported by providers on-chain as part of the reward proof. They are cryptographically protected and tamper resistant.

Under Passable QoS, providers reach a minimum or 'passable' threshold to maintain full rewards on the network, providers are rewarded in direct proportion to the quality of services rendered, and because of the cryptographic proofs involved, neither consumers nor providers are able to profitably give false quality reports. For consumers, there are no rebates or refunds. Every relay costs 100% of the CU, with providers guaranteed 50% of the potential reward. For providers, bad quality of service decreases actual profitability (rewards earned) without affecting potential profitability (rewards-to-be-earned). The system works virtuously with consumers bearing the cost of their relays and providers bearing the cost of their service.

5.3.2 QoS Excellence

QoS Excellence is an accumulative algorithmic consumer-side scoring mechanism which aggregates provider performance scores across multiple sessions with multiple consumers. Its goal is to encourage providers on the network to exceed operable standards and excel relative to their peers. QoS Excellence is calculated from latency, availability, and sync scores from each provider combined with existing QoS Excellence scores in a time-decay function. A provider's score for QoS Excellence is calculated thus only considering the current score and the given score as well as the time elapsed, where the half-life is approximately 1 hour. Preference is thus given to the most recent scores in each category which best represent the reputation of the provider in question.

Scores for each of the categories are computed against a benchmark which must be excelled to get favorable scores. The exact means of computation for each category is detailed in the table below:

Criteria	**Scoring**	**Metric**
Latency Score	The mathematical ratio of the raw latency to the following: (retry_time/2) where ‘retry_time‘ is the time until another provider is engaged, set in the specification, or defaulting to CU*100ms for each API call in the session.	Raw time measurement / Retry Time Benchmark
Sync	The mathematical ratio of the average block time to the following: the time difference from when the same block number was provided by a different provider, plus the block gap time multiplied by the average block time.	Time gap measurement / Average Block Time Benchmark
Availability	The mathematical ratio of successful relays to total relays in a given session	Successful Relays / Total Relays

Table 6: Table of Criteria for QoS Excellence Scores per Relay

Scores are never combined but entered into a decay function. The function works as an averaging algorithm across all scores received by a provider from its serviced consumers. QoS Excellence scores are sent on the same relay request as QoS Passible scores but as a separate field. Each time a consumer scores a provider, their score in a category updates that provider's score. The exact details of how these accumulated scores for each category are arrived at are mathematically advanced, but a simple update formula can be represented below:

$$\text{update Formula:} \quad \frac{\text{current average num} \times \text{time\_decay}(\delta t) + \text{latest\_sample num} \times \text{weight}}{\text{current average denom} \times \text{time\_decay}(\delta t) + \text{latest\_sample denom} \times \text{weight}}$$

Each score received is utilized in other calculations performed by the protocol to influence pairings such as those used by the Pairing Engine, in the Provider Optimizer, and in Smart Consistency mechanisms. As such, the QoS Excellence ratings can be seen as a provider's time-adjusted reputation for service in each of the three service categories. It is a massively influential and important part of the protocol - as it is the reputational system for providers across the entire network.

Under QoS Excellence providers receive more pairings (i.e. more chances to profit) based upon prior performance, a provider can stake less for the same level of selection favor, if they have better service, and the scoring of consumers on a given provider's performance accumulates with additional service.

5.3.3 Clustering

Clusters are groups of consumers with similar plans, experience levels, and service quality history. Clustering has two distinct goals - the first is to prevent attacks where a coordinated group of consumers collude to disproportionately affect targeted providers; the second is to enhance service delivery for consumers with similar preferences and history.

Quality of Service scores for each provider are distinct for each cluster. Thus, a provider may maintain a Quality of Service score amongst one set of consumers that is reflected differently in another set of consumers. Consequently, amongst a single cluster, a similar effect is observed in PairingList formation, but not provider selection via the Provider Optimizer. Clusters give consumers the opportunity to pair based on provider performance for similarly profiled consumers (peers). The end result is statistically better pairings and more successful relays.

6 Performance Management

6.1 Uptime

Uptime is the capacity for the network to forgo service disruption and remain available for service delivery. In a distributed data network such as a blockchain, high availability is a highly desirable trait [13]. Lava protocol implements multiple measures to ensure maximum availability and prevent predictable sources of service disruption from resulting in fragmented or broken service delivery including constant availability, response retries, and health checks.

6.1.1 Constant Availability

Constant availability is the capacity of Lava network to keep service delivery even when the blockchain is no longer available. When the blockchain is unavailable, consumers and providers already familiar with each other can agree to still conduct relays - even if such relays exceed the normal limits that are in place. As a result chain halts do not result in service disruption. After the network has been down for a configurable duration, relay payment transactions accrue while peer-to-peer connections between providers and consumers continue in "virtual epochs." Consequently, data exchange continues despite any halt in the Lava blockchain. The `x/downtime` module is used to identify if blocks stop advancing. If such an event occurs, exchanges continue, new payments are stored in queue for resumption, and providers and consumers go into constant availability mode. Upon resumption, the queue is settled automatically and the rewards system accommodates unclaimed rewards. With separation of blockchain advancement and RPC queries, consumers and providers can still query based on the previous state of the blockchain. Thus, PairingLists can still be generated from the current non-advancing state.

6.1.2 Response Retries

Relays do not always succeed; relay requests do not always return valid data. On Lava protocol, responses have four possible results:

1. **Protocol errors** result in immediate retries.
2. **Node errors** are returned to users.
3. **Valid responses** are returned to users.
4. **Hanging responses** are measured against recommended retry time either set in the specification or calculated from compute units for the API method. If the time is exceeded, the consumer sends the relays to an additional provider.

In the event that a response returned is invalid or experiences a time out, the protocol automatically initiates retries until a response is received. A state machine manages interaction with multiple providers to ensure that at least one provider returns a valid response. Availability is significantly boosted for consumers on the network who experience built-in fallback in case an individual provider errs to give a valid response.

The above flow describes the protocol's handling of light queries. The protocol implements different retry strategies based upon the API - for sake of nonce consistency, light queries, heavy queries, and transactions are handled differently.

6.1.3 Health Checks

The consumer client is designed to periodically monitor and update the health status of providers at specified time intervals. Periodically, the consumer checks the health of providers with a mechanism called probing. Probes are a minimally-sized gRPC message that have no on-chain fees, do not convey data, and are sent over an existing connection. When probing is activated, the consumer asynchronously checks a provider's latency and synchronization/freshness and orders the provider's endpoints by their assessed health. It also disqualifies providers which are not answering probes - triggering "jailing" by reporting the provider as unresponsive (the exact means of jailing is defined in a later section of this paper). The interval at which health checks are executed can be configured or disabled at will. Probe results affect the provider optimizer's selection process. The upside of this is that no relay latency is wasted on establishing providers' health. It's especially meaningful when a transition is made to a new group of providers on an epoch change, by mitigating disruptions to existing relay traffic.

6.2 Integrity

Integrity is the capacity for the network to return honest, fresh, and accurate data to requests. Lava protocol contains many mechanisms designed to preserve network integrity and data reliability. Integrity is enforced on the protocol via the use of smart consistency and conflict detection and resolution measures.

6.2.1 Smart Consistency

Smart Consistency is a mechanism that solves common block inconsistency and mempool inconsistency issues. As implemented, it mitigates the impact of differences in latest block and nonce state across nodes. Commonly used consistency solutions introduce tradeoffs [14]. Smart consistency does not rely on commonly used forms of stickiness which are used ubiquitously in web3. For mempool inconsistency, this is accomplished by forcibly propagating nonce state to multiple providers on each transaction. For block inconsistency, this is accomplished by calculating the probability that a provider has the correct block and selecting among only providers in which this is highly likely.

Stickiness refers to the characteristic of a system or process where certain elements, such as users, tasks, or data, tend to remain within a particular node or server. This concept is often utilized in load balancing and resource allocation strategies. In relation to blockchain, it is the likelihood for a consumer to return to the same provider given a different relay. Using Stickiness solves both mempool and block inconsistency. However, Stickiness has advantages and disadvantages that are less than the optimal trade-offs possible. Smart consistency is a more effective alternative.

Block inconsistency refers to the situation where multiple nodes have reached different states in the blockchain. The situation can result from network delays, soft forks, software issues, or malicious activities. Working across multiple nodes who may be synced to different blocks on the blockchain will produce inconsistent or unavailable answers. To solve block inconsistency, three steps are taken:

1. the consumer calculates the chance a provider has the needed block and algorithmically determines the provider most likely to be synced to the correct spot, The combination of the consumer's last seen block, most recently requested block, as well as the selected provider's latest block, and the latest block of all available providers are used to create a score. Based upon this score, the consumer is able to select a provider who they reasonably believe will be able to answer their relays with the freshest block.
2. The provider calculates the likelihood of serving the correct data and disqualifies itself if it cannot reasonably do so. The provider process is hard-coded to evaluate whether it can plausibly return the necessary data before supplying the consumer with a response. To calculate the chance of success, an inverse upper bound gamma regularized function is used:

$$P(X \leq k) = 1 - \frac{\Gamma(k+1, \lambda)}{\Gamma(k+1)} \tag{1}$$

   This function is far more precise than taking an arithmetic average - yielding a precise moving average for the last 25 blocks. Based on the aforementioned calculations, the provider returns an error if the block gap, that is the distance between the consumer's requested block and the provider's latest block is too great to be overcome.
3. The consumer verifies the finalization proof in the headers of the provider response to ensure that the data is indeed correct and has not been misrepresented or corrupted. This finalization proof (contained in the headers of a relay response) also expresses to the consumer what the provider's latest block is. A consumer has the ability to check the hashes and ensure that a provider is telling the truth.

Mempool inconsistency is the state of nonces differing that can result in confused provider responses. This is particularly noteworthy on requests that affect nonces and the visibility of transactions. Such requests easily fail when accessing the mempool of a different provider than expected. Mempool consistency is established using request propagagtion. Whenever a transaction or query based on the local state is encountered, the provider sends the transaction or query to all the active/available providers in the same Pairing list of that epoch. This ensures that each consumer's usable providers experience the same state transformation at the same time and eliminates possible discrepancies. This also brings many benefits to consumers as externalities:

1. Forcefully propagating mempool data increases the chance that a transaction gets to a validator faster and thus creates latency resistance for the network.
2. Enforcing duplicated state across multiple independent actors prevents the possibility of censorship.

Smart Consistency mitigates blockchain inconsistencies by utilizing accountability measures, message parsing, and metadata analysis on a relay-by-relay basis to do predictions on which providers are most likely to provide the appropriate response. The end result is that a given consumer's usage of paired providers can be inclusive of more providers without stickiness while maintaining data integrity.

6.3 Conflict Detection & Resolution

Conflicts are detected by the protocol and resolved in an on-chain resolution 'jury' mechanism. Conflict detection is performed consumer-side, via cross-verification of deterministic responses across multiple providers on identical queries. If responses conflict and are correctly signed by multiple providers, consumers can report potentially fraudulent providers with a conflict transaction. Providers are selected as jurors and must help adjudicate or face penalties. Providers identified as fraudulent by this process are penalized. The first step to ensuring data integrity is conflict detection. Conflict detection is the recognition of inaccurate data. Each consumer client verifies the integrity of data independently. This approach contrasts with the common approach of specifying designated entities ("fishermen") to sample providers. A fisherman can be isolated through collusion - providers run by the same entity can identify the fisherman by its pattern of queries and game the system to avoid detection. By relegating detection to each consumer client, a provider has no easy way of avoiding the conflict detection mechanism.

Performing detection tasks on every query would be computationally expensive. To avoid this expense, consumers choose between two different modes of detection:

- **Statistical** - the consumer verifies a percentage of requests against other providers
- **Per Query** - the consumer verifies each query (as identified by a flag).

Additionally, detection introduces trade-offs between data integrity and delivery speed. Thus consumers can choose between two strategies for each query:

- **Blocking** - queries are not returned until a provider has checked data and verified no conflict exists. When integrity is an absolute necessity, blocking queries guarantee data is checked and valid data is returned. However, on some requests this can take significantly longer to return a response.
- **Optimistic** - queries are returned to a consumer as soon as available, conflicts are detected later and reported on the chain after the fact. Because the data is already returned by the time of detection, a user cannot be protected from the data. However, on most requests this approach will be significantly faster.

Consumers test data by sending the same requests to multiple providers. Providers must prove a response is accountable. In order to be eligible for proof, a response meets three criteria:

- **Deterministic** - the data to be verified must be deterministic (i.e. non-random; query produces same outputs given the same inputs). Non-deterministic queries will produce different answers on the same query and create a false positive conflict detection.
- **On The Correct Fork** - the data must be on the same fork of the blockchain in question. The same deterministic query, on the same specification, sent to providers on different forks will produce a different answer and create a false positive conflict detection.
- **Finalized** - the data must be finalized on the blockchain in question. Non-finalized data can be subject to chain reorganization and produce discrepancies resulting in false positive conflict detection. If a reorganization does happen and both providers are on the same incorrect fork, other providers might not be able to vote.

Accepting the prior criteria, a request R will have a provider response (X_1) which is equivalent to any given provider response (X_n) if and only if the consumers' request is deterministic (D), on the same fork (F), and returns finalized data (G). This is represented logically below:

$$\forall R\,(D \land F \land G) \to (X_1 = X_n)$$

There are three types of conflicts which a consumer can report:

1. *Finalization Conflict* - Finalization conflict detection uses finalization data to prevent providers from lying on latest block metadata in replies. Providers are incentivized to give the latest block possible by Quality of Service measures so there must be some way to prevent them from cheating. Providers commit on finalized block hashes. Proofs for finalized blocks are compared by consumers.
2. *Response Conflict* - Reply conflict detection uses response data to prevent providers from lying on deterministic and finalized data within responses - mismatched data from different providers is detected when consumers compare responses.
3. *Same-Provider Conflict* - Same-Provider conflict detection uses block hashes to determine if a provider is inconsistent. Consumers detect these conflicts by checking if the same provider gives two responses on the same query which have different block hashes, yet are finalized.

If a consumer detects conflict in any of the three scenarios, they bundle the providers and their hashed responses in a conflict detection tx and report them on chain:

lavap tx conflict detection [finalization-conflict] [response-conflict] [same-provider-conflict] [flags]

Once reported, the protocol takes two steps. First, validation involves verifying the conflict (checking the signatures of all parties, ensuring the data is different, and checking the specification was the same). This step is trivial and involves simple verifications. Second, resolution involves resolving the correct response and rewarding and punishing providers accordingly. This step is critical and explained below.

Conflict resolution is handled via jury commit and reveal, a double-phased secret vote process. A subset of providers are selected at random as a 'jury.' The jury votes on the correct answer by providing a hash of the answered original query with an added salt. Initially, the identities of voters are known, but the content of votes is unknown. Once all provider responses are received, reveal of the vote takes place. The providers share salts and responses, revealing their identities. All votes of providers are counted based on stake-weight: provider stake plus all delegations. Responses being deterministic, from the same spot on the same fork, and finalized means that, even in the worst scenarios, the correct responses should be represented by the majority of providers who return a response. The `x/conflict` module defines the exact majority percentage to use.

To incentivize good provider behavior, the minority provider(s) gets disciplined; the majority providers get rewards. The provider with the fewest votes, and any members of the jury which voted with them, are penalized by having a fraction of their staked tokens taken and distributed among all the providers who voted correctly and the provider who gave the original correct response. If the jury voted incorrectly, providers that calculated a different vote outside of the jury can make a deposit to increase the jury size. If the vote changes they are rewarded, however if not then they lose their deposit.

6.4 Scale

Scale is the capacity for the network to handle an increasing volume of consumers, providers, and relays.

6.4.1 Fixed-size Relay Payments

Proofs of payment are aggregated and submitted together. In the current implementation, all relays are aggregated per session and in the future, sessions within the same epoch will be aggregated together via signer or using zero-knowledge methods. Each sequentially submitted transaction contains signed proof of the number of CU expended to provide its data. There is no merkle tree of responses that a provider must provide. The end result is that the size of relay payment data does not increase dramatically with increased relays.

6.4.2 Retroactive settlement

Payment to providers is retroactively settled. Lava employs a 'lazy blockchain design' with relatively large block sizes which allows providers to submit claims for rewards after the passing of an epoch. Providers do not need to receive rewards instantly, but can opt to make requests for payment when gas prices are low or when otherwise convenient.

6.4.3 Multi-session Management

Sessions happen simultaneously without interference. Both provider and consumer can concurrently interact with multiple parties. Because data exchanges happen off-chain and only proofs of relays are posted on chain, the blockchain structure imposes no bottleneck on the throughput of relays. Furthermore, the protocol supports unordered transactions by consumers by using un-incremented session-ids. Session-ids are randomly chosen uint64 values. Although messages are sent sequentially to the provider, there are no collisions (due to unique session ids) and no double spend (due to relying on session ids) instead of nonces.

6.4.4 Dual Caching

Both providers and consumers employ an optional caching service to decrease network latency and improve network scale. Provider process and consumer client cache recent calls on a cache service external to itself. The cache service can be hosted on a completely discrete machine and significantly increase the rate of service. The mechanism which implements caching is sophisticated and takes into account synchronization of data and consistency to ensure fresh data is returned. The cache service uses the local RAM of the machine it runs on for the latest blocks, and creates local databases for finalized older data. When caching services are enabled and a call is made which is likely held in cache, the relay can be returned significantly faster than ordinary.

6.5 Selection

Selection is the capacity for consumers on the network to exercise choice over pairing. Selection is a sought after amenity which enhances performance for preferential consumers.

6.5.1 Provider Optimizer

The provider optimizer is a module on the consumer client which algorithmically decides, given a set of QoS scores and a selection strategy, which provider in a consumer's Pairing List will receive a given relay. The provider optimizer does not affect pairing, but is active after pairing to ensure real-time selection of the best provider available to a consumer within an epoch. The provider optimizer uses an involved algorithm to compare the last seen block of a consumer with the last known block of a provider in order to determine whether a provider can service the relay request.

Every relay attempt begins with the provider optimizer going over all providers on a consumer PL for the given epoch and returning a ranked subset of providers. It does this by iterating through each provider's address and analyzing their Quality of Service Excellence (reputation) for sync and latency - weighting them according to the rules of the consumer's selection strategy. A perturbation factor (also affected by selection strategy) is added to each score in order to introduce some randomness. The best provider is placed in the '0' index of the returned providers list - ranked the highest - and is sent the relay(s).

Aside from using selection strategies to direct relays to the most optimal candidates, the provider optimizer does two very important functions:

1. **Introduces Random Perturbation** - unless a strategy selects against it, introduces random perturbation to a provider selection, following a normal (Gaussian) distribution, with the scale of perturbation determined by the original value and a perturbation percentage set by strategy.
2. **Mitigates Probability of Error** - uses the Provider's complete reputational information, the current block, and the consumer's request to mathematically determine the probability of a successful relay:

$$\begin{aligned}
P_{\text{blockError}} &= \text{CalculateProbabilityOfBlockError}(\text{requestedBlock}, \text{providerData}) \\
P_{\text{timeout}} &= \text{CalculateProbabilityOfTimeout}(\text{providerData.Availability}) \\
P_{\text{success}} &= (1 - P_{\text{blockError}}) \times (1 - P_{\text{timeout}})
\end{aligned}$$

The provider optimizer is an essential part of how consumers select providers on the protocol.

6.5.2 Selection Strategies

Selection strategies are employed by the provider optimizer to decide which provider is most optimal to a given consumer. Selection strategies work by assigning weight to different aspects of the Quality of Service Excellence score. A consumer uses a balanced strategy by default. On a balanced strategy, latency is preferential (0.6 weight) - however, strategies are configurable and multi-dimensional - meaning a consumer can express multiple preferences and shift weights accordingly.

The provider optimizer uses the following strategies:

Strategy	**Description**
STRATEGY_BALANCED	Default weights; may the best scores win
STRATEGY_LATENCY	Preference fastest providers; best latency scores
STRATEGY_SYNC_FRESHNESS	Preference providers with most-up-to-date data; best sync scores
STRATEGY_COST	Always pick a single provider per relay
STRATEGY_PRIVACY	Distribute relays across more providers; decrease stickiness
STRATEGY_ACCURACY	Select multiple providers and get quorum
STRATEGY_DISTRIBUTED	Increase randomness, reduce weight of reputation

Table 7: Provider Selection Strategies

Strategies are implemented by affecting variables in the Provider Optimzer's algorithm such as the `perturbation percentage`, `exploration chance`, and `latency weight` and ultimately result in a different chosen provider for relays.

7 Economic Incentives

Economic rewards exist to align incentives amongst actors on the network. Economic incentives use the native token of the network, LAVA (subsection 2.3.4). LAVA token is used as a means to pay for gas fees required for transactions, to purchase subscriptions for consumers, as a governance token allowing holders to participate in network decisions, and as rewards distributed to validators, providers, champions, and delegators. Lava has a fixed supply of tokens with no inflation. The total amount of coins will never exceed $10^9 \times 10^6 ulava$ or 1,000,000,000 LAVA.

7.1 Incentivized Public RPC Pools

Incentivized Public RPC Pools are rewards pools funded by participating ipRPC blockchains and sponsors. ipRPC pools may hold Lava's native token (LAVA) and IBC wrapped tokens of any kind in a module account. Rewards are distributed to providers from ipRPC pools alongside the normal rewards received by providers in the network, discussed in the following section.

Incentivized pools rely on a Server Kit / gateway interface for access and connectivity. Elected consumers, with special subscriptions, act as a sponsor to end users. These special subscriptions enable the consumers to distribute rewards from the ipRPC pool to providers when end-users consume data. As a result, providers who serve the specification for ipRPC receive rewards over and above normal service rewards on the network.

Prominent members of ecosystems such as DAOs, Foundations, and major contributors fund the module account and set the parameters for rewards. The accounts are funded by special transactions of any token and a fixed amount of LAVA tokens. Any IBC-compatible token, including LAVA, works [4]. The rewards structure has a configurable emission schedule. However, the funds must cover the minimum cost, a parameter determined by governance. The minimum cost prevents spam and establishes a protection against valueless tokens being used to incentivize service in Lava - by ensuring that a minimum threshold of value is distributed to providers. With ipRPC pools, end users get RPC at no cost, offloading expenses to an authenticated consumer, while providers receive additional rewards from the network.

7.2 Payments and Pricing

7.2.1 Provider relay rewards from subscriptions

Providers receive base rewards in proportion to two main factors: total CU from consumers served and Passable QoS scores. However, the total amount of consumers served can be affected by many factors and so there are many indirect influences on provider rewards; these include provider stake, consumer clustering, and Quality of Service Excellence scores. As stated, these secondary factors do not directly affect calculable rewards, but exhibit indirect effects by influencing the likelihood of pairing.

7.3 Calculation of Rewards based on CU

The provider process uses a CUTracker to keep records of serviced CUs by a provider for each specific subscription. The CUTracker is an on-chain ledger with records of all CUs serviced on the network. Whenever a provider delivers service, the number of CUs associated with the relay is counted and submitted to be saved on-chain by the CUTracker. The CUtracker is reset monthly for providers that service a subscription.

Provider payments are derived from CU and paid in LAVA. Each provider receives a portion of a serviced consumer's subscription price. The portion received of a subscription is derived by dividing the amount of CU a provider serviced for that consumer by the total amount of CU a consumer used in a subscription. This gives the exact percentage of serviced CU which is attributable to the provider in question. Used CU is tracked per specification per consumer, and not accumulated across different specs. A simplified formula, demonstrating the relationship between CU and payment, which does not incorporate Passable QoS (subsection 5.3.1) (table 5) or other fees is shown here:

$$\text{Payment} = \frac{\text{subscription\_price} \times \text{provider\_tracked\_CU}}{\text{total\_CU\_used\_by\_subscription}} \tag{2}$$

7.4 Claiming Rewards

Providers are entitled to claim rewards based on the computational units (CUs) used, which are automatically recorded via transactions sent by the service after each epoch concludes. These CUs are tracked for a month from the start of a subscription or its monthly anniversary. Subsequently, providers receive their respective shares of the rewards. Once allocated, these rewards are distributed among their delegations, including self-delegation, and can then be claimed through a transaction. It's important to note that rewards are retained on-chain until they are claimed; providers face no penalties or time limits on claiming their rewards. In the future, the protocol may implement further aggregation of sessions to optimize block space and reduce gas fees. This would involve combining multiple live sessions into a single transaction.

7.5 Staking and Delegation Rewards

7.5.1 Validators

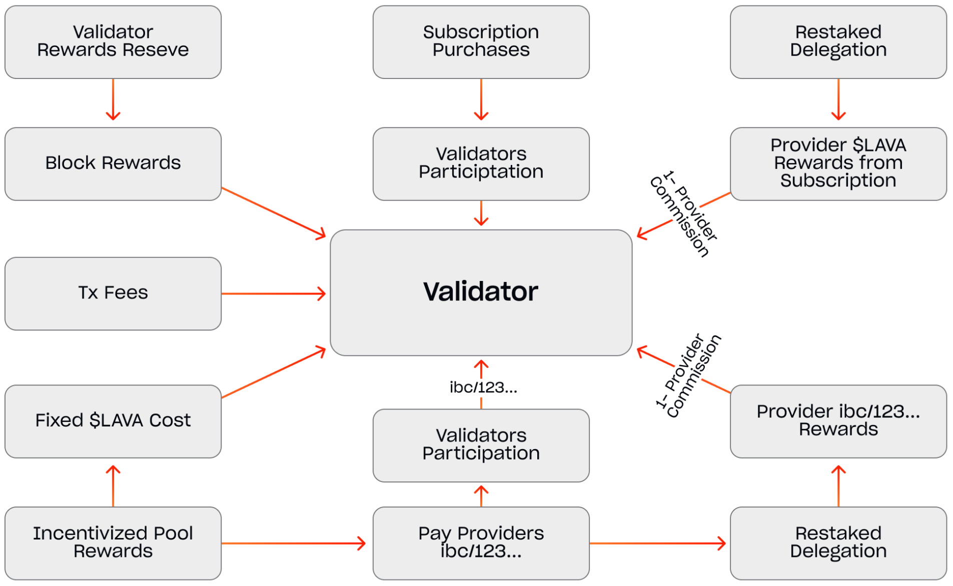

Figure 4: Validators absorb rewards from multiple sources

As in other blockchains, validators stake on-chain to earn rewards. Staked validators get rewards from four sources: 1) Block rewards from the Rewards Reserve, 2) Block rewards from subscription commission, 3) Block fees from transactions on the network, and 4) Delegation rewards (as described below in Restaking).

Block rewards work similarly to other Cosmos blockchains, but have some key differences due to the constraints imposed by the protocol's fixed supply of tokens. On most Cosmos blockchains, the rewards for the block are created with the block. However, Lava network premints all rewards tokens and stores them in a Rewards Reserve.

Validator rewards are drawn from the validator allocation pool in the Rewards Reserve. Tokens are allocated monthly from the preminted supply in the Rewards Reserve into a `validators_distribution_pool`. The amount that enters the distribution pool varies based upon block times. Faster block times induce lower pool balance whereas slower block times encourage a higher pool balance. The token balance of the `validator_distribution_pool` is always a fixed sum equal to or less than the total validator rewards earned in a given month; because Lava has a fixed token supply and no new tokens are minted to reward validators, validator rewards are only claimable from the available pool balance.

A `bonded_target_factor`, set by governance, also affects rewards. The `bonded_target_factor` reduces the amount of bonded tokens by validators by adjusting validator's rewards and thus altering their incentive to stake. If too much LAVA is staked by validators, the factor can be decreased; if too little LAVA is staked by providers, the factor can be increased to ensure full rewards are given. This prevents validator rewards from hampering liquidity on the chain. The equation below depicts the relationship:

$$\text{Reward} = \frac{\text{validators\_distribution\_pool\_balance} \cdot \text{bonded\_target\_factor}}{\text{remaining\_blocks\_until\_next\_emission}} \tag{3}$$

At the end of each month, any remainder of validator distribution pool balance which is not awarded to validators for service is burned.

7.5.2 Providers

Like validators, providers must stake on chain to be eligible for rewards. However, contrary to validators, providers earn rewards from: 1) subscription payouts from serviced consumers, 2) ipRPC rewards for serviced ipRPC specifications - as described above in ipRPC pools (subsection 7.1), 3) Delegation Rewards - as described below in Restaking (subsection 7.6) and 4) Rewards Boosts (subsection 7.7).

Subscription payouts form the base compensation for providers on the network. The base compensation for providers is calculated by taking each consumer's subscription cost, calculating how much a provider contributed to the CU on a subscription for a given consumer on the rewards period and then subtracting the nominal fee percentage taken for the DAO and validator distribution. A simple formula is shown here:

Provider i's base compensation =

$$\sum_{\text{Consumer } j} \frac{j\text{'s subscription cost} \cdot \left(\frac{\text{CU provided by } i \text{ to } j}{j\text{'s total CU used}}\right)}{(1 - \text{validators distribution\%} - \text{DAO\%})}$$

The reality is that there are many factors that determine a provider's participation in rewards, beyond what is described in base compensation. Factors such as the amount that others have delegated to a provider affect compensation. A provider's participation in service rewards is also a function of the proportion of total stake for which they are solely responsible:

$$\text{provider\_participation\_in\_reward} = \frac{\text{subscription\_reward} \times \text{provider\_stake}}{\text{effective\_stake}} + \frac{\text{commission\_percentage} \times (\text{effective\_stake} - \text{provider\_stake}) \times \text{subscription\_reward}}{\text{effective\_stake}}$$

7.6 Restaking

Restaking is a novel concept for earning rewards on the network controlled by the `x/dualstaking` module. The purpose of restaking:

- Gives delegators a way to help select the best providers and earn rewards for it
- Provides additional yield for validator delegators
- Accomplishes lower security fees on the network

With restaking, token holders can delegate their tokens to a specification served by a provider to claim a portion of rewards awarded to the selected provider. All restakers are eligible for a portion of the profits of that provider.

LAVA delegations to Validators can be restaked. LAVA tokens can be restaked to providers without introducing additional collateral, earning more rewards at the cost of higher risks. In the reverse, whenever a provider stakes tokens, an equal amount is restaked to a validator. In other words, a provider stake is both self-delegation (to oneself as a provider) and standard delegation (to a validator).

Optionally, when a validator stakes tokens, an equal amount can be staked to a provider. To support the legacy Cosmos `x/staking` module [15], which does not have provider delegation, an empty provider is used by the dualstaking module, as a default placeholder. After a successful initial stake, validators can redelegate from the empty provider placeholder to a selected provider on the network - earning rewards as the provider services consumers.

7.7 Reward boosts

During the early stages of Mainnet, demand may not be sufficient to bootstrap the launch of certain APIs and services. The protocol's rewards module employs a pre-allocated pool of tokens to enhance network rewards. Providers who participate early and service requests reap the benefits as they are paid out from this pre-allocated pool. Reward boosts scale proportionately with demand, up to a specified cap. If the cap is reached, the boosts diminish until they are burned. This ensures that rewards never create an inflationary effect on the token supply and introduce undesirable economic incentives.

Additional rewards are given for servicing those which have been determined to provide a bonus by governance. Specifications with a lot of staked providers offer bonuses for providers who service them. These are "shares" defined in the specification, which is accepted via an act of governance. Shares act as a multiplier which can be used to make servicing a specification more valuable. For default bonus rewards, an equation is provided below:

$$\text{ProviderBonusRewards} = \text{TotalSpecPayout} \cdot \frac{\sum_{\text{payment } i}(\text{provider base rewards}_{i,j} \times \text{adjustment}_{i,j})}{\sum_{\text{provider } j}\sum_{\text{payment } i}(\text{provider base rewards}_{i,j})} \tag{4}$$

In the above equation, `adjustment` accounts for how much the consumer used other providers. Adjustment is a multiplier between 1 and 5 which adjusts the bonus rewards based upon the spread of providers - it is a collusion protection that prevents bonus rewards from being reaped by consumers who favor a smaller sampling of providers in a naive attempt to farm payments. Provider bonus rewards are supplementary to provider base compensation and offer another form of economic incentive for servicing in-demand APIs.

7.8 Slashing/Jailing Conditions

Normal slashing of validators is handled through the existing modules on CosmosSDK [16]. However, Lava implements both slashing and jailing for providers on the network. Slashing and jailing function as economic disincentives to malicious, faulty, or inconsistent providers. Through these mechanisms, providers which are unavailable or unreliable reduce or lose the ability to reap rewards from network participation.

Slashing is the process of decrementing provider stake. It happens under exceptional circumstances, either when a provider has been proven to be the source of an illegitimate response in on-chain conflict resolution or when a majority governance vote takes aim at a provider's stake. Conditions that lead to slashing include the following: Returning data that is provably incorrect or false Returning stale data

A governance proposal which specifies the provider address, chain ID, slash factor and the proposal deposit is sufficient to enact a slashing event. The slash factor is the percentage of coins to be slashed if accepted. If the proposal passes, the provider is unstaked completely and will get $\text{original\_stake} \times (1 - \text{slash\_factor})$ coins after the pre-defined unbonding time has passed. All of the provider's delegations will be slashed by the same percentage. The delegation will still exist (even though the provider is unstaked) so the applicable delegators can redelegate to another provider or wait for the provider to restake.

Jailing is the process of disqualifying providers from pairing. Providers who are jailed do not receive requests beginning from the next epoch after they are jailed. While jailed, a provider will retain their stake, but will not appear in pairing lists or receive rewards. Conditions that lead to jailing include the following: Returning consecutive errors on data relay requests Returning no response to relay requests Attempting to respond to relay requests with an unsecured connection

Unresponsive providers are disclosed in consumer relays with responsive providers. Once an unresponsive provider has been contacted, the consumer moves to another provider on the Pairing List. After reaching a responsive provider, the consumer client reports the addresses of the previous unavailable provider(s) and the amount of CU of the requests, respectively. Then, every epoch for each provider, if the complainer's reported CU is larger than the serviced CU over a constant number of epochs, the provider is jailed.

7.9 Champions On-Chain Rewards

Each specification details a `Contributor` and `ContributorPercentage`. `Contributor` is actually a list of Champions, listed by Lava public address, who are responsible for the creation and/or maintenance of the spec in question. `ContributorPercentage` indicates the percentage of coins the champion will get from making the contribution. This is essentially a commission on the usage of the specification that rewards the contributor for all data exchanges that take place from providers employing the specification on the network.

8 Beyond RPC - further use cases for Lava

Lava is a network of data providers adding, serving and scaling different kinds of services. The Lava protocol ensures accountable and performant peer-to-peer communication, with no single point of failure. As such, Lava can bring modular quality of service and scale to any service added to the network. Below are some of the exciting possibilities for Lava to expand towards:

- **Unified and decentralized RPC access**: Lava's first use case is decentralized RPC. Lava creates unified access points such as endpoints available from a Gateway or providers available in common web3 SDKs.
- **Indexing solutions**: Specs can be added for basic RPC and more specialized APIs across all chains. Indexers return data and use Lava blockchain as a settlement layer for rewards, while end-users enjoy service delivery guarantees.
- **Large Language Models & AI queries**
  - With the increase in computationally advanced language models the use of server resources and advanced trained models will increase significantly
  - With many models available, the ability to interact with several suppliers simultaneously and pick the best result is a competitive advantage; Lava enables this use-case. AI queries can be routed through the network to competing LLMs by the self-same means as RPC requests.
- **Intent solvers**: An intent is a transaction that does not specify execution but rather specifies result: ex. 100 ETH for 50 LAVA, with a spread: +- LAVA. If consumers send an intent over Lava, multiple providers can solve it, attach the batch of transactions used to do so, and then send it back to a Lava client. The Lava client can verify it and if veritable send a transaction over Lava to fulfill the intent. If there are intent verifiers, they can independently run verifications on the solution and provide feedback
- **Cross chain decentralized tx bundling** (eg `ETH_TX_BUNDLE`): Validators on target chains (eg. `ETH`) can set up bundling endpoints in Lava. They would prove ownership by signing with their Validator key, while keeping their RPC bundles private. This way only Validators are accessible to transactions over this spec. The Validator performs the bundling, and when the block is created with the bundle they return the block to the user, ultimately getting paid in LAVA.
  - A user can also send to only a specific trusted validator to bundle, and as a result reduce exposure of the bundle to the bare minimum before block inclusion (a longer timeout)
- **Light Peer-to-Peer Client**: With the decrease of active nodes and the increase of light client/node high quality solutions, the load on nodes from light nodes is increasing. With no monetization of P2P load, the service for light clients is expected to deteriorate. Lava offers the possibility of a high quality p2p access to nodes with exclusivity and higher rate limit + lower latency (better QoS). Dedicated nodes will expose their p2p over Lava for light nodes to use over Lava
- **Decentralized Sequencers** - Currently, a lot of rollups utilize a centralized sequencer; all user transactions need to get to that sequencer in order to be included. Several sequencers may join Lava and provide block creation services. Block creators (or sequencers by turn) can take these blocks, and settle them on the Lava Blockchain.
- **Secure Wallets/Nodes** - Secure wallets require high resources to run solutions that utilize AI and statistics to identify malicious transactions. These solutions require a user to reroute his transactions to these parties. Lava offers the opportunity to seamlessly send the transaction to multiple secure wallets
- **Decentralized Oracles** - Lava allows adding a modular specification that supports alternative web3 data functionality such as Oracles. Instead of dealing with lock-in for a specific oracle, consumers can query the network and receive answers from the most responsive and most available oracle services.
- **Seed Accountability** - Currently seed nodes are altruistic or foundation funded, and they have no accountability on their service neither in uptime nor data. Instead of managing seed nodes through social means and trust, seed nodes can join Lava, and receive incentives from the ecosystems. Lava protocol enforces service accountability, reducing malicious attacks without touching the seed node or the client code.

9 Conclusion

We have presented Lava Network, a modular network designed to be the access layer for all chains and rollups, starting with RPC/APIs.

- Blockchains use Lava to scale and optimize their data infrastructure, giving developers and users reliable access to their ecosystem
- Developers and users send data requests to a decentralized network of data providers, routed based on geolocation, historical quality of service (QoS) and stake
- Providers join the network to earn native tokens of chains on Lava, or to earn LAVA tokens
- Delegators can restake to providers, helping boost the selection frequency of top providers, and improving QoS for all users
- Beyond RPC, rollups can add many types of data services to be optimized through Lava such as decentralized oracles, indexing and sequencing
- Lava's vision is to unlock permissionless innovation, by making it easy for blockchain developers to onboard dapp developers and users

A Appendix

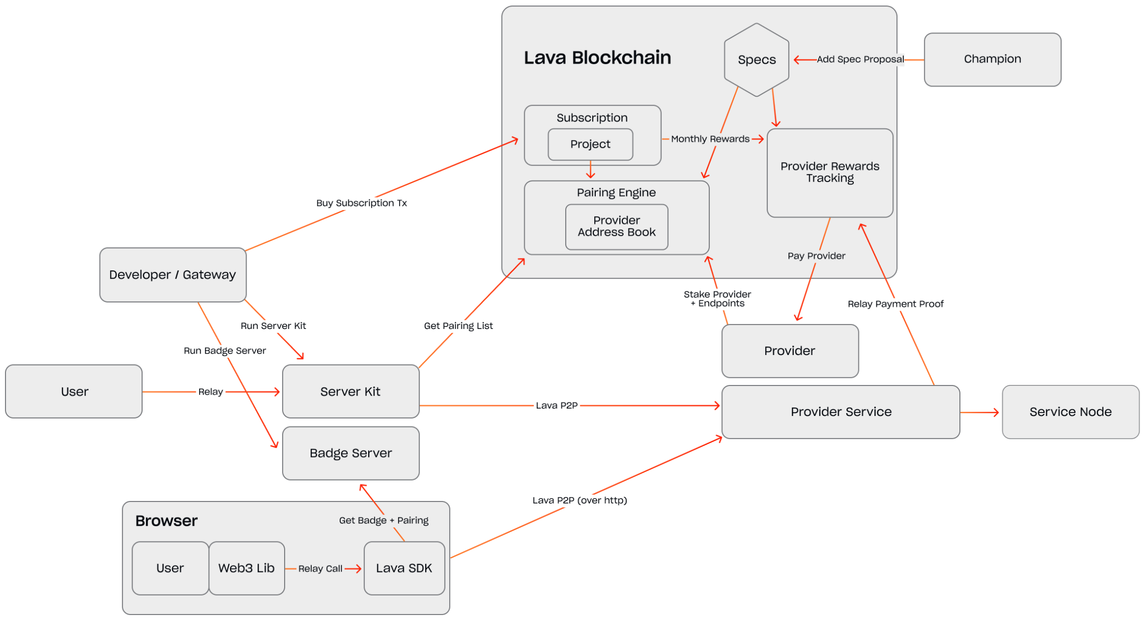

Figure 5: A detailed overview of the protocol; outlining all areas of interaction as discussed throughout the paper

References

[1] Moxie Marlinspike. My first impressions of web3. https://moxie.org/2022/01/07/web3-first-impressions.html, January 2022.

[2] Osato Avan-Nomayo and Vishal Chawla. Hackers hijack ankr gateway for polygon and fantom networks. https://www.theblock.co/post/155413/hackers-hijack-ankr-gateway-for-polygon-and-fantom-networks, July 2022. HACKS.

[3] Osato Avan-Nomayo. Infura and alchemy are now blocking access to tornado cash. https://www.theblock.co/post/162402/infura-and-alchemy-blocking-access-to-tornado-cash, August 2022. INFRASTRUCTURE.

[4] Cosmos Network. Ibc token transfer. CosmosNetworkTutorials, n.d.

[5] W. Li, X. Deng, J. Liu, Z. Yu, and X. Lou. Delegated proof of stake consensus mechanism based on community discovery and credit incentive. *Entropy*, 25(9):1320, 2023.

[6] Microsoft. Remote procedure call (rpc). https://learn.microsoft.com/en-us/windows/win32/rpc/rpc-start-page, February 2022.

[7] Cosmos Network. Governance module. https://docs.cosmos.network/v0.50/build/modules/gov, n.d.

[8] Cosmos Network. Bech32 addresses. https://docs.cosmos.network/v0.50/build/spec/addresses/bech32, n.d.

[9] Google Developers. grpc. https://grpc.io/, n.d.

[10] Bellsofaba. Understanding farcaster: A sufficiently decentralized social graph protocol. https://medium.com/@bellsofaba/understanding-farcaster-a-sufficiently-decentralized-social-graph-protocol-838d078f5067, 2023.

[11] S. Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.

[12] C. Moore and S. Mertens. Interaction and pseudorandomness. In *The Nature of Computation*. Oxford Academic, 2011.

[13] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, 2002.

[14] IBM. What is the cap theorem? https://www.ibm.com/topics/cap-theorem, n.d.

[15] Cosmos Network. Staking module. https://docs.cosmos.network/v0.50/build/modules/staking, n.d.

[16] Cosmos Network. Slashing module. https://docs.cosmos.network/v0.50/build/modules/slashing, n.d.