THE ROOT INFRASTRUCTURE FRAMEWORK

V 1.22

RIF Gateways

Building the infrastructure for the next generation of distributed applications

Contents

Contents 1

Abstract 2

Introduction 2
- Consuming external-world data 2
- Consuming blockchain data 4
- Scheduled delegation of transactions 5

The problem 6

Design 7
- **Data services** 7
  - External data update-rate 8
  - Consumption models 8
    - Pull model 8
    - Subscription model 8
      - Subscriptions governed by SLA contracts 9
    - Dispute resolution 9
    - Payment models 9
    - Data Service example 10
  - Architecture overview 10
- **Trigger services** 11
  - Pre-defined triggers 12
  - Custom triggers 12
  - Consumption models 13
    - Pull model 13
    - Subscription model 13
      - Subscriptions governed by SLA contracts 13
      - Dispute resolution 13
  - Architecture overview 14
- **Transaction scheduling services** 15
  - Consumption models 15
    - Pull model 15
    - Subscription model 15
      - Subscriptions governed by SLA contracts 16
      - Dispute resolution 16
      - Gas Price 16

Appendix: RIF Gateways applications 17
Escrows 17
Conclusions **18**
References **18**

Abstract

Blockchain protocols with on-chain smart-contracts must communicate with external systems through Oracles; while external systems might need to trigger certain actions based on events emitted from the Blockchain. There's also the necessity of crucial time-based functionality such as blockchain transaction scheduling and, specifically for contracts, recurrent function invocation.
RIF Gateways provides an implementation-agnostic protocol for all these types of data-related services.

Introduction

Consuming external-world data

Smart contracts are digital agreements enforced by computer code. Several instantiations of the smart-contract concept have been devised: as an auditable but centralized trusted-computer everyone connects to, as a consensus-enforced execution in a decentralized network of nodes, or as an off-chain market-based scheme that incentivizes honest execution. In this paper, we focus on consensus-based smart contracts in which the code executes on a resource-constrained virtual machine that runs programs written in a Turing-complete language, and it's executed in a decentralized environment, such as the RSK or Ethereum blockchains.

The blockchain and its decentralized nature make smart contracts tamper-proof since no party can alter any contracts' code, execution input, or results. As a consequence, smart contracts provide a new level of trust, eliminating the need for intermediaries and generating a more secure system for digital agreements.

Smart contracts often require external data to fulfill their core operations. Some applications that typically require external data are the following:

- *Insurance contracts* (e.g., Etherisc) need information about external events and pay whenever a specific chain of events takes place.
- *Trading contracts* (e.g., Synthetix) need access to price feeds, for example, the USD/EUR exchange rate.
- DeFi crypto borrow & lending contracts
- *Sports betting contracts* need information about game results to evaluate bids and award winners.

Since smart-contract execution must be deterministic, on-chain smart contracts present a drawback compared to off-chain counterparts, because they cannot access data that is not already in the blockchain.

Smart-contract logic must be executed using the same input parameters and data, even when they require information from an external data source. The execution of the contract becomes susceptible to manipulation from an external attacker when the data does not only depend on input parameters but on other factors, such as time of day, BTC to USD exchange value, the temperature in a specific location, the request for a random number, etc.
A possible solution would be a service that retrieves data based on a query ID associated with certain input values. This service would ensure that all the requester nodes of a specific query ID get the same value and would guarantee a deterministic result in their verifications. The problem with this "solution", which relies on a trusted source, is that a flawed source can compromise the security and trust-minimized model of blockchain applications. A data source owner could intentionally manipulate the offered data in order to damage a particular consumer or break consensus.
Also, fetching the data from a unique source puts entire trust on that single third party, regardless of any measure taken to prove that the data has not been tampered with during transit from provider to requester.

Without solving the consensus-enforced data problem, many real-world applications that rely on external data cannot be implemented using smart contracts.

Therefore, there arises a need for access to real-world data feeds in the blockchain. This need must be satisfied by a secure, tamper-proof and trust-minimized solution that guarantees a deterministic output (all miner nodes must get the same value for a specific query request) on values fetched from external sources.

Solutions to tackle this problem have been devised, and the mechanism by which external data is brought on-chain is called "Oracle". Examples of existing solutions ("Oracle Services") are the following:

- Centralized (e.g., Provable). A centralized Oracle solution that fetches data from a user-specified data source, along with a demonstration that it is genuine (the data-source used is the one the user requested) and untampered.
- Decentralized (e.g., Chainlink). An Oracle network with an off-chain or on-chain aggregation of results originated from different Oracle provider nodes. These nodes can be selected manually or automatically based on a reputation system, and their respective query results are submitted for aggregation using a commit/reveal scheme.
- Prediction Market (e.g., Augur). A decentralized prediction market where users can ask or bet on the outcome of future events. Oracles are the individuals that, as randomly-chosen reputation token owners, are responsible for settling markets (collectively select the outcome of a question), based on the number of reputation tokens that they have staked on a particular outcome. In conclusion, the outcome of a future event that gets submitted to the blockchain is based on the "Wisdom of the crowd" phenomenon and crypto-economic game theory.

Consuming blockchain data

Off-chain applications might need access to data and events originated within the Blockchain in order to complete their logic and operations. Some examples of applications that typically require access to blockchain data are the following:

- *Game industry,* where games must process smart contract events in order to show updates to the participants.
- *Domain Name Services* (e.g., RIF Name service), which require users to be notified in case of domain license expirations.
- *Payment Services* (e.g., RIF Lumino), which require alerts and notifications anytime a payment channel is opened or closed.

Currently, there are different applications and tools to monitor what is happening inside a blockchain, for example, a block explorer. But these types of solutions do not provide an easy way to listen and capture specific events, nor an API or developer support for integration (e.g., libraries). Another option is to configure a blockchain node to directly observe state changes, even though it implies a technical background from the user or programmer about smart contracts and their state, on-chain events, and also maintaining a node instance.
The best solution is to develop an event processing service that allows any user to configure what they want to listen, what actions to take once that event happens, and even subscribe to already-defined templates.
This solution is called "Trigger". Examples of existing solutions are the following:

- Web Services (e.g. Hookpad). The event processing service hooks to an EVM-compatible blockchain and lets the user listen and filter events.

- Customized VM (e.g. Tenderly) next to synchronized nodes. An agent connects to any ethereum-compatible node (Quorum, Geth, Parity, etc.), does light processing of the executed transaction info, and sends it to a system containing a custom EVM which re-executes the transaction. The custom EVM can trigger alerts/notifications not only based on smart contract events but on any kind of smart contract activity (e.g, certain parameters were passed).
- Broadcast domain. Instead of the user configuring their own event hooks, it proposes a subscription-to-events service which through a message bus, broadcasts each event and lets the users consume them through that bus. One implementation over Ethereum is Eventeum, which proposes a decentralized event/transaction listener with a REST interface to subscribe/unsubscribe to new events and pluggable broadcast mechanisms.

Scheduled delegation of transactions

Blockchain is quickly becoming the new software platform. As such, users of this platform expect crucial time-based functionality such as transaction scheduling.

Transaction scheduling has some movement already:

- Centralized: This solution requires the user to run her own blockchain node client. The client would hold the transaction in memory until the desired time arrived (e.g., Parity UI, now deprecated)

- Decentralized: An off-chain network of TimeKeepers/Delegated Executors that claim schedule-transaction requests generated in the blockchain and perform their delayed execution in exchange of a bounty fee (e.g., rework of the Ethereum Alarm Clock protocol made by ChronoLogic)

The importance of a decentralized protocol is clear, it avoids having a single point of failure (e.g, The centralized piece of software responsible for executing the transaction at a certain point in time disconnects) and it’s also censorship-resistant (a centralized scheduler may purposely skip the scheduled transaction execution).

In this work, we present a new RIF Service for the consumption of external data sources, internal blockchain events, and scheduling of transactions. This service proposes an interface layer for unified access to these data services.

The problem

User smart contracts, also called internal consumers in this document, need a way to retrieve external (off-chain) data from service providers to be used within their blockchain solutions.

External consumers need to retrieve data from the blockchain to be used within their solutions. In addition, external consumers need a standardized way to consume blockchain events without the effort of implementing a new solution for each use case.

All of them also need an affordable method for retrieving data periodically, with predictable fees based on the usage requirements negotiated with the data service provider and with termination clauses they can execute in case of an unacceptable quality of service. In the case of internal consumers, this is of utmost importance in order to have an effective recurrent function call implementation.

Currently, there's no data service marketplace available. Users cannot search and look for data services, read user reviews or compare data service details of different providers (e.g., data sources consulted, subscription models offered, data aggregation methods applied, Oracle technologies in use, usage fees). Instead, the user needs to know precisely which data source to use and implement a solution around a third-party Oracle implementation to consume such data.

In order to consume external-world data, the user must know how to use these Oracle solutions, have strong confidence in the reputation of the data sources she wants to consume, and perform the extra work of estimating costs for querying each source of data.
Extrapolating to a use case where many types of data are required in a smart contract clearly evidences the lack of usability. In the worst-case scenario, it may also discourage the use of external-world data, generating a negative impact on the adoption of new and exciting use cases that depend on this kind of information.

A similar problem arises when trying to listen for specific events. For the simplest solution (only listening for thrown events), the user needs to know beforehand the address of the contract and the Event name for each specific notification she wants to add in her solution. Users cannot subscribe to an already implemented solution via micro-payments (e.g. using RIF Lumino) or create through a user-friendly interface her own monitor. Instead, external users need to involve in technical tasks like downloading and running a blockchain node (or consuming from one, or better yet, several) and implementing a new monitoring solution or

trusting in a monitoring service provided by a third party, thus adding a point of centralization.

Time-based transaction execution, (scheduling a function call inside a contract or from an external application) is not a natively-supported feature in any blockchain. A solution to offer it as a simple-to-use decentralized service is imperative.

Design

RIF Data Gateways service defines a simple interface for consumers to interact with Data Service Providers, Trigger Services, and Transaction Scheduling.

Data services

Currently, some Oracle solutions exist and differ in many aspects such as:

- The way they resolve a query. For example, only from one data source or aggregating many of them (on-chain or off-chain).
- The pre-defined data sources they provide. For example, a secure HTTP query to retrieve data from an API or random value, Computational Knowledge Engine (such as Wolfram Alpha), content stored in the InterPlanetary File System (IPFS), the resolution of an arbitrary computation in a secure enclave, or prediction markets.
- Centralized vs. Decentralized.
- Open source vs. private.
- Payment method options for the data retrieved.

A **data service** provides a specific kind of external-world data (which could come from a single data source or an aggregation of multiple data sources). The logic required to fetch this data is implemented in a smart contract. The Service Provider is the creator and offerer of this data service.
Consumers can choose among different types of data services and then interact with the corresponding Data Service Provider's smart contract for getting the external data. Each Data Service Provider contract may be associated with a unique domain name (e.g., *price.bitcoin.rif*) which makes them discoverable by human-readable IDs. The domain name and type-of-service can be registered on the RIF Marketplace.
In order to comply with the Data Service protocol, Providers need to implement the Data Service interfaces in their smart contracts. RIF Data Services protocol is designed to be integrated with third-party Oracle Services.

External data update-rate

Depending on the use case, a consumer may need data that is the latest published version or that is not older than a certain period of time. Therefore, data may be updated periodically by the provider. For example, a smart contract may need to know the USD/BTC exchange rate. Since USD/BTC value is highly volatile, having the latest available rate may be desired. On the other hand, a consumer that needs to know a country's long-term interest rate may tolerate using a value fetched one hour ago, since this kind of information is not statistically volatile on a one-hour time frame.

Consumption models

There are two ways for an on-chain consumer to interact with a provider. A consumer can subscribe to a data service paying a subscription fee, or just directly pay-and-pull new data on an as-needed basis.

Pull model

In the *Pull Model,* a consumer pays and retrieves the data on a per-query basis. The requested value is retrieved directly from the data source. Data must be fresh, so no caches are intended to be used, and this type of service model is expected to be more expensive compared to others (and slower, since it implies the data request plus a callback invocation once the data is available).

Subscription model

In the Subscription model, a consumer pays a fixed price for access to the data, which may be cached by the service provider and updated only periodically. Serving a single piece of data to multiple customers allows the Service Provider to split the cost of fetching the external-world data among all the subscribers (resulting in a more accessible service), and to offer even better prices depending on the periodicity or "resolution" of the data offered.

RIF Data Services protocol proposes two simple subscription modes for consuming external data:

- On-demand: the consumer asks the Data Service Provider for the value on an as-needed basis, as long as the subscription is valid.
- Push: the Data Service Provider pushes the new data to the subscribers periodically.

Subscriptions governed by SLA contracts

Consumers may specify the requirements they expect from the service in a new Service Level Agreement (SLA) contract, and Providers can offer their matching data services by registering to this contract.
Also, consumers can search for a service provider (by domain name or any other keyword) using RIF Marketplace and choose from one of the data service options the Provider is currently offering. The provider will generate an SLA contract and the consumer will agree to the terms by registering to this contract.

Dispute resolution

Consumers expect to receive correct values every time they fetch new data from a Provider or when they receive new data from a callback invocation (in the subscription model). They also expect the Provider to comply with the update-rate defined in the SLA.
If any of these requirements are not met, the consumers may initiate a dispute. Dispute resolution implementations may be on-chain or off-chain.
The quickest and cheapest procedure would be first trying an off-chain dispute resolution. The consumer would send the evidence (off-chain) to the Service Provider or emit an event the provider would be listening to. The provider would then process the customer's dispute request off-chain and pay the customer if necessary. In case of not being able to resolve the dispute, the customer still has the on-chain process.
For on-chain implementations, the dispute resolution logic may be added to the SLA contract, but another solution is to use a decentralized, incentive-driven, arbitrating platform similar to what Kleros offers.
In a decentralized dispute resolution system, jurors will vote on one of the available resolution options; the description and behavior of each option will be entirely defined by the SLA.
The resolution of a dispute may include, among other options, the cancellation of the subscription, a fee discount or even a penalization fee.

Payment models

Payments for any service can be performed using RBTC or ERC-677 tokens, and multiple payment models may be supported. The subscription model may support standard token transfer, payment channels like Lumino or on-chain probabilistic payments, whereas the pull model may implement on-chain payments using the "withdraw pattern". Additional research will be conducted to determine the cost and benefit of each payment method.

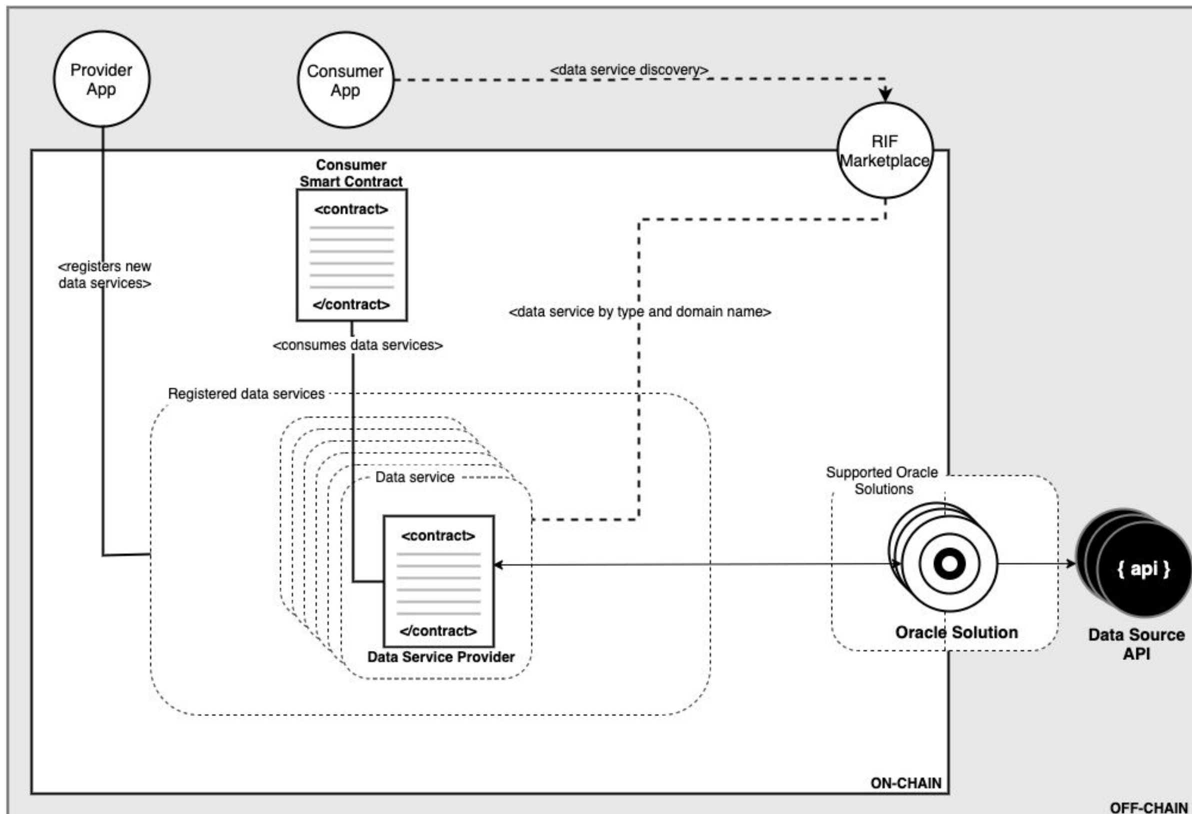
Data Service example

A vendor wants to offer a data service that provides the BTC price in different currencies. It generates a Data Service Provider smart contract and registers it under the domain name "*vendor1.price.bitcoin.rif*" using the RIF Name Service (the provider may also specify a service type and additional metadata).
The generated smart contract includes injected logic supporting both Pull and Subscription models, it also uses https://blockchain.info/ticker as the external data source and the initial centralized Oracle Service implementation offered by RSK.
Users will be able to discover *vendor1.price.bitcoin.rif* using any third-party app that feeds from the RIF Marketplace API. Since this data service supports both consumption models, users may pull the data just once or opt for a subscription.
The vendor may also proactively offer *vendor1.price.bitcoin.rif* as a BTC price solution to customers currently requiring (and asking for) this kind of data service.

Another vendor may offer a similar service but with a wider selection of data sources whose values get aggregated and then returned to the user.
This vendor generates a different Data Service Provider smart contract and registers it under the domain name "*vendor2.price.bitcoin.rif*".
It also offers both Pull and Subscription models, but instead of just using a predefined external data source, it requests several BTC price values from a decentralized (off-chain) Oracle network. The final value may be aggregated off-chain or on-chain, depending on the Oracle solution implementation.

Generic data services would be possible as well, for example, a vendor could provide generic secure HTTPS REST services under "*https.rest.rif*".

Architecture overview



Trigger services

A **trigger service** provides information from within the blockchain (which could come from a single data source or an aggregation of multiple sources). The logic needed to fetch the data could be implemented off-chain. The Trigger Provider is the offerer of this service.

By using the APIs exposed by the Trigger Provider, a consumer could build her own notification solution on top of it (e.g., SMS, e-mail).

Each Trigger Provider should be associated with a unique domain name (e.g., *yourkittie.cryptokitties.trigger.rif*) to make it easier for users to interact with it. Additionally, Trigger Providers need to comply with a predefined interface to allow users to discover and filter the offerings through an explorer application (e.g., filter by price, number of notifications in a given time period, etc.). This interface should be designed to be consumed by third-party applications.

Once a consumer chooses a provider for the events she is interested in, she can engage with it following the provider's consumption model (either pay for a fixed amount of events or for a fixed period of time) and start listening for on-chain events.

Instead of using a single Trigger Provider, consumers can opt to subscribe to a set of Trigger Providers and perform aggregation of the notifications received.

Dispute resolution details could be handled by an SLA contract similarly to what happens in the Data Services

Currently, some Trigger solutions exist and differ in some aspects such as:

- The way the trigger consumes and processes blockchain information (e.g., a node(s) consumer or a customized VM).
- Centralized vs. Decentralized.

Pre-defined triggers

The Trigger Provider may offer a notification service about particular smart contracts or VM events (e.g. Events of frequently used smart contracts such as ERC20, ERC721, etc. are usually interesting for the user).

In this setting, the Trigger Provider notifies a fixed set of events emitted by the contract being observed, which may be already defined by the provider. Finally, the service may expose a mechanism to let the user apply filters to the set of events being notified.
Using as an example the RIF Token, the list of events would be "Transfer" and "Approved", the predefined contract would be the RIF smart contract address, and a possible filter would be the recipient's address.

Custom triggers

The Trigger Provider allows consumers to build a trigger for their own needs. It provides a simple interface that allows a non-technical-user to create her own notification service. The costs of it could be pre-defined or negotiated between parties.

The interface exposed by Providers should let consumers specify the source of the events to be notified (e.g. consumer could provide a smart contract address). Also, the trigger lets the consumer set the list of events which will be notified by the Trigger Provider and parameterized conditions (e.g., an event name, a transaction with a specific hash, transactions with a specific "from" or "to" address, etc.). Finally, to trigger some action, the Provider

allows consumers to specify which action is executed once a matching event is received (e.g. send a request to a given endpoint, send an email, etc.).

Consumption models

The trigger service offers the pull and subscription models

Pull model

A consumer requests a single event notification (the most useful scenario is when listening for something that happens only once, like a specific transaction hash, but could also be used when just one notification is needed) and pays the required amount. When the pre-established conditions are met, the provider will execute the required action, and then discard the condition listener.

Subscription model

As with Data Services, a consumer pays a predefined price for the service (which can be renewed). Depending on the trigger implementation (e.g., Tenderly re-executes the transaction in their custom VM), listening for a specific event and offering it to multiple customers might allow the Service Provider to split part of the cost of processing the conditions among all the subscribers (resulting in a slightly more accessible service).
For triggers we only have the **push** subscription model; the Service Provider executes the required action when the pre-established conditions are met, while the subscription is valid.

Subscriptions governed by SLA contracts

As in the case of Data Services, consumers may specify the requirements they expect from the Trigger Provider by using a Service Level Agreement (SLA) contract and Providers can offer their matching trigger services by registering to this contract.

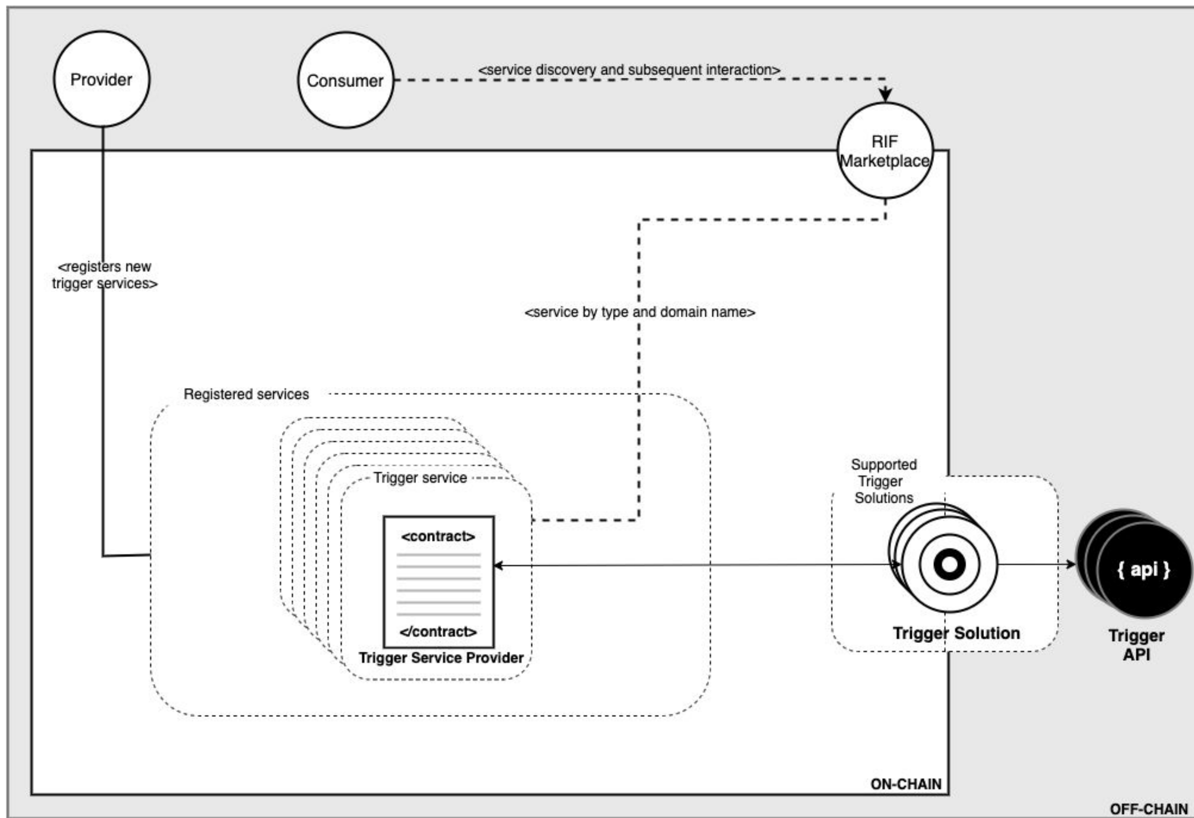
Dispute resolution

The same mechanism described for Data Services applies for the Trigger Services. Consumers expect the Provider to comply with the terms defined in the SLA. If any of these requirements are not met, the consumers may initiate a dispute.
Trigger Services need to generate proofs about the service they provide (e.g., if a notification was emitted when the specific conditions were met) to foster off-chain dispute resolution. These proofs should have at least the following properties:

- Be uniquely associated with a particular notification/event.
- Include evidence of the actual invocation.
- Be time stamped.
- Be as small as possible.

- Be tamper-proof (can't be generated at time-of-dispute)

Architecture overview



Transaction scheduling services

The transaction scheduling service is a decentralized solution that allows a customer to program future executions of on-chain transactions.
Customers may be internal/on-chain (a smart contract), where the most functional use case for requesting a scheduled execution is the recurrent invocation of a transaction (most likely, a recurrent oracle invocation), or external/off-chain, where, for example, a value-transfer transaction for a specific time or if a given condition is met is the most likely scenario.

The logic required to request a scheduled transaction execution is orchestrated by a set of smart contracts and our initial proposed service is based on the delegated execution pattern and similar to the chronos protocol that Chronologic has defined in their whitepaper.

As with Data Services and Trigger Services, new scheduling service providers can join by registering a new scheduler service that will be discovered through the RIF Marketplace.

Consumption models

The scheduler service can also offer the pull and subscription models, but their definition differs from the data service model

Pull model

A consumer requests a single transaction schedule for a delegated execution and pays the required amount to ensure it's future execution. The execution can be scheduled for a specific time and a given "execution window" (window time to actually execute the transaction, starting from the given absolute time)

Subscription model

Here the definition diverges a bit from data services, where we had a subscription to a specific data service, the analogous here must be the subscription to delegate the execution of a specific function in a recurrent manner.

The consumer pays a negotiated price for executing a function recurrently (e.g., every five minutes, every 3 blocks, when a condition is true, etc).
Here we do not have the benefit of saving money as happens with Data Services, where a provider of a popular service can save money by having multiple subscribers of a single data request. A transaction schedule is a delegated execution and it's the customer's execution, there's no shared execution scheme.

The only variable here is the periodicity of the function call, and the benefit of the subscription is to standardize the recurrent call feature.

RIF Scheduler Services protocol proposes just the push subscription mode for delegating a recurrent execution (there's no "on-demand" execution of a transaction scheduled to be executed with a pre-established frequency)

Subscriptions governed by SLA contracts

As with the previous services, consumers may specify the requirements they expect from the Scheduler Provider by using a Service Level Agreement (SLA) contract and Providers can offer their matching scheduling services by registering to this contract.

Dispute resolution

The same mechanism described for Data Services applies for the Scheduling Services. Consumers expect the Provider to comply with the terms defined in the SLA. If any of these requirements are not met, the consumers may initiate a dispute.
Scheduler Services need to generate proofs about the service they provide, which is simpler than the proofs required in the Trigger Service, since the conditions for executing a transaction are known, and the expected output evidence can be found in the blockchain (the execution of a specific transaction) to foster off-chain dispute resolution.

Gas Price

A scheduled transaction is still a blockchain transaction, regardless of the scheduler implementation. A transaction is sent to a blockchain node and then forwarded to other nodes. These nodes will add the transaction in their transaction pool only if their acceptance criteria is met (usually it just depends on the gas price put in such transaction).
The transaction pool will, most likely, be sorted by gas price so the higher the gas price the more likely the transaction will be added to the next block to be mined by that node.
The pool is also finite, there's always a chance for the transaction (with low gas price) to be discarded from the miner's transaction pool.
In the scenario of a scheduled transaction with a poorly estimated gas price, a mechanism should exist to minimize the chances of it being discarded by the miners:
The mechanism to use will depend on the Scheduler implementation.

The most likely Scheduler solution would be based on Chronologic, and consists on having a Transaction Request contract that will be executed by a Time Node when required. At that moment the Time Node can select a new gas price if needed. The Time node is an off-chain

node in charge of accepting transaction-schedule requests (based on the Time Node's conditions) and submitting it to the blockchain when the requested conditions are met. That execution is actually invoking an specific method of the Transaction Request contract previously mentioned. There's a new Transaction Request contract per transaction schedule requested.

The gas price solution analyzed by Chronologic as "the best" is putting a minimum gas price in the scheduled transaction request. At execution time, that value will be reimbursed to Time Node , but if there's a gas price spike, the Time Node will then select a new value (to ensure the transaction is added in a block in the expected time frame) and pay that difference (losing some rewards but not withholding a transaction).

If the solution is different, for example, with a scheduler that submits the original transaction in the specified time, then a different solution is needed. The user should be able to contact the scheduler and update the scheduled transaction (so it has a higher gas price).

Appendix: RIF Gateways applications

In this document we presented three services, the first service is for consuming external data from the blockchain (Data Services), the second one is for consuming, from the external world, data from within the blockchain (Triggers Service), and the third service is for requesting future execution of a blockchain transaction (Scheduler Service).
RIF Gateways may also involve solutions that interact in both ways with the blockchain. For such solutions, there should be interfaces which allow the non-technical-user to create her own gateway application in support of better blockchain adoption. One example of an on-chain solution that needs external information and might need to communicate with the external world is the Escrow Service.

Escrows

In the blockchain ecosystem, there are a vast number of services being promoted today. Some of them might require the intervention of a third party to achieve an agreement among the different parties involved. An escrow is a legal concept in which financial instruments or assets are held by a third party on behalf of two or more parties that are in the process of completing a transaction. The assets are held by the escrow until contractual obligations have been met. Users are motivated to use escrow services they can rely on, in this sense, trust becomes a relevant aspect of a well-architected solution. A decentralized blockchain escrow service (e.g one of the use cases of Kleros) for dispute resolution is the logical option for a

user to transact in the blockchain ecosystem for services, products or assets with a high level of confidence.

An Escrow Provider Service allows the parties involved in a dispute the creation of decentralized escrows. The service exposes an interface which allows each party to lock funds, describe the conditions that must be met, and watch smart-contract or off-chain data to ensure that the conditions were achieved.

Conclusions

RIF Gateways leverages the existing research on blockchain Oracles, Triggers, and Scheduling solutions by abstracting and extending all this knowledge into a single unified API. The API allows users to discover gateway services, consume them directly or via subscription models, and to pay the required consumption fees using Tokens or cryptocurrency.
Negotiated terms and conditions are governed by a Service Level Agreement that both the consumer and provider must sign.
Service providers can use any of the available Oracle, Trigger, or Scheduler implementations. New service implementations can be added.
RIF Labs will continue to research data Oracles, Triggers, and Schedulers. Current research topics include cost optimization by using ephemeral data, payment channels, off-chain probabilistic payments, and multiple data queries in a single on-chain request.

References

[RIF Payments]
https://www.rifos.org/payments

[RIF Data Gateways Interfaces]
https://www.rifos.org/gateways

[RIF Name Service]
https://github.com/rnsdomains/

[Ephemeral data]
https://github.com/rsksmart/RSKIPs/blob/master/IPs/RSKIP28.md

[Navite on-chain Probabilistic Payments]
https://github.com/rsksmart/RSKIPs/blob/master/IPs/RSKIP55.md

[Kleros] Clement Lesaege and Federico Ast. (2018)
https://kleros.io/assets/whitepaper.pdf