Userfeeds Documentation

Release 0.1.0

Grzegorz Kapkowski

May 24, 2018

Contents

CHAPTER 1

Introduction to the Userfeeds Platform

Userfeeds is envisioned as an infrastructure platform which allows developers to utilize easily content (popularity) rankings for their own use or for their clients and build similar ones whenever needed.

The connection with the Userfeeds Platform, which can be used for that purpose, is exectuted through the http of any *API* which allows to contact with the platform, to get data from and about internet transactions as well as content rankings, and to access databases in which data on transactions, which make the rankings, are stored.

Internet transactions are blocks of the Javascript code which can be called code snippets and placed on websites as elements serving for engaging visitors into promotional activities (such as liking and promoting) which are based on the use of crypto-currencies, mainly Ethereum and Bitcoin. Such transactions are technically called *'Claims'*. The *Claim* is a basic data entity in the Userfeeds Platform. It may represent an "endorsement", a "like", an "upvote" for a given URL / Text / Identifier which is signed by one of the supported signing methods. You can *sign* a *Claim* with the ECDSA and send it through the HTTP or make an Ethereum transaction with data of *Claim* which then will be treated as signed.

Rankings for transactions can be obtained through the utilization of algorithms which are actually pre-defined queries designed for searching through data aggregated in Ethereum and Bitcoin blockchains. Those queries can also be seen as a set of tools supporting the interface. Examples of content (popularity) rankings are: a list of products on amazon.com, songs in a playlist on spotify, news on NYTimes, and all other places which have some content (links, articles, songs) sorted in a particular way which you can access.

As a matter of fact it is possible to see in ranking (popularity) reports how scores have been made, who and in what way has influenced the score, information on users who have made the score in a selected perspective.

The platform is also capable of tracking financial information (records) regarding particular users and transactions via smart contracts which are a sort of agents defined for information obtaining and storing.

In order to make full use of the *API* and all algorithms it is essential that you have a wallet and an account for the Etherum crypto-currency created. Currently Ethereum, Bitcoin and IPDB are supported on the platform.

For easy set-up of both the wallet and the account, the MetaMask service can be used available at https://metamask.io/.

For more information on crypto-currencies, visit topic-related websites which serve information on how Ethereum and Bitcoin work and are valuated.

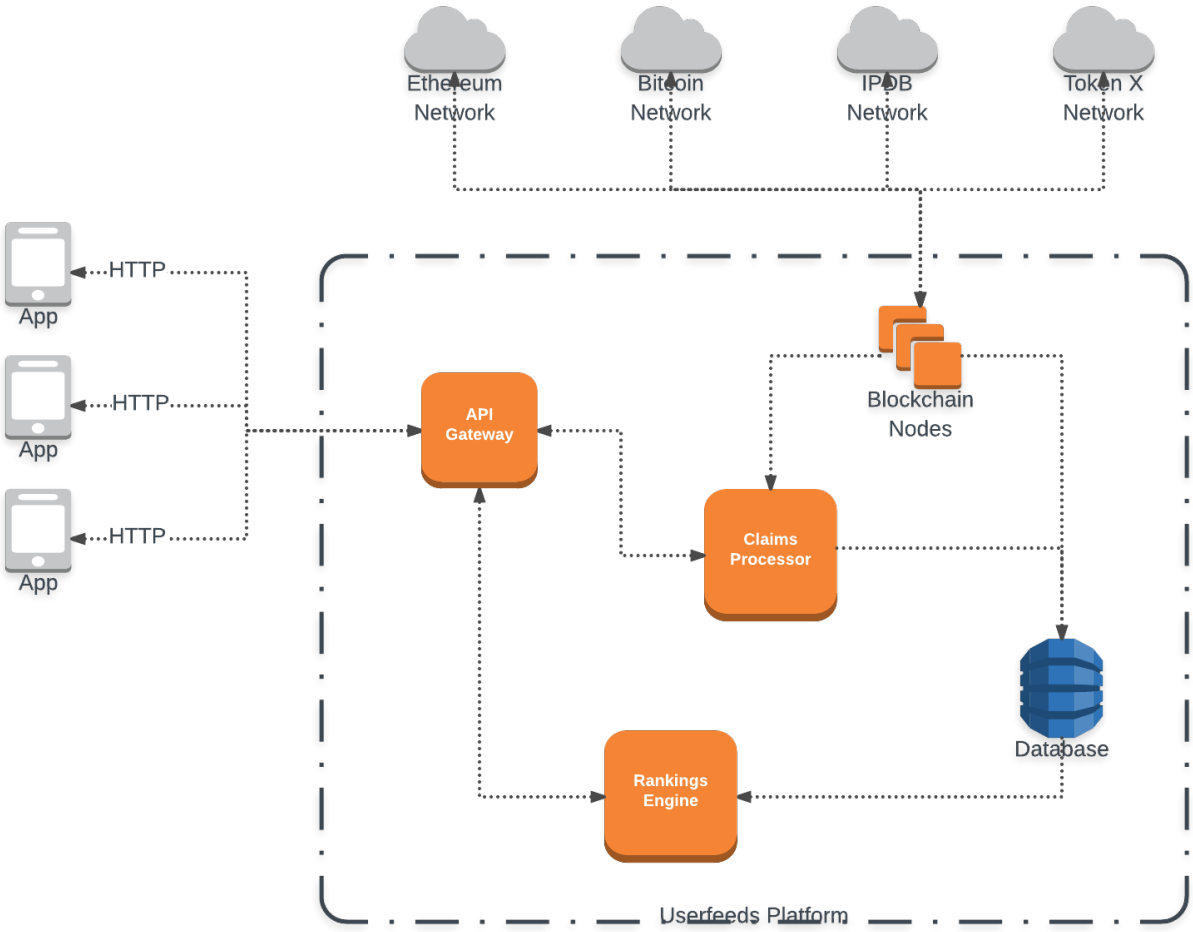
In order to become familiar technically with the Userfeeds Platform and be able to use it efficiently, go to *Quick Start - Guide for Developers* and *API Reference* to see how to use our 'read-only' APIs and how to use the Ethereum

blockchain for pushing information into the Userfeeds Platform. Later on you can go to http://api.userfeeds.io/portal/, and register as a developer, and go to http://api.userfeeds.io/portal/apis/ API Catalog to request for an API Key. When the API Key is ready for you - start using all of our HTTP API's.

Our mission - as owners of the Userfeeds Platform - is to help developers in obtaining information on populatiry (rankings) and reliable scores (reports on aggregated transaction data per needed perspective) as well as earning money for introducing promotional actions on web sites by providing them with 'ready-to-use' API's, libraries and algorithms which they can incorporate in an easy way into their software and web pages.

1.1 How does the Userfeeds Platform work?

The Userfeeds Platform is composed of a couple of parts which interact together in order to deliver rankings through *HTTP API's* to your application.



It is important to remember that any internet transaction is a *claim*. Such *Claim* is a basic information package signed cryptographically (with use of unique key) and connected with an account - in the json format - key - claim / key - target.

The structure of a *claim* can be presented in the following way:

from who ; **for whom** ; **how much / amount** ; **transaction identifier**.

The *claim* can also be understood as metadata for any transaction.

Claims created with help of our algorithms land on the Ethereum blockchain from where they are collected and read. *Claims* can have extensions which make them a particular type.

In order to learn more about claims, go to claims

First it is the Userfeeds Platform which reads all the information from supported blockchains and stores it in an internal database. We store all transactions and contract calls in the Graph database and current balances in the standard SQL database.

When someone sends *Claim* data to the Userfeeds Platform through the HTTP or through the Ethereum network, it is processed and inserted into the Graph database for further use in ranking algorithms.

When an application makes a HTTP request to our Ranking API endpoint, a Ranking Engine takes the requested algorithm and applies it to the current data stored in the Graph database and the SQL database and returns sorted entries for application to display to the user.

1.2 Why use the Userfeeds Platform?

- All our algorithms are of Open Source type and therefore can be improved any time when an issue arises or changed per current needs
- Everyone is able to create their own custom rankings
- We use the blockchain as the source of ranking signals, for example - how many tokens are connected to given content(s), how stable were token holders endorsing given content(s), how involved they are in given token(s).
- Everyone can monetize their application / site easily by providing *sponsored* ranking on their application / site
- Everyone can deliver superior experience for their users and visitors by customizing our ranking output basing on your needs
- Introduction of promotional elements and activities to any web service is very easy with our algorithms for web components
- Everyone can make their blogs more engaging by bringing widgets for promoting
- Everyone can benefit from products and services of our partners

CHAPTER 2

Quick Start - Guide for Developers

2.1 How to get all links connected with Ethereum

The code snippets presented below the Demo window return a list of news or messages shared to an Ethereum context and sorted according to our *simple* algorithms.

The obtained information and rankings shown in the Demo window below are a result of applying the algorithms presented per programming language in sections beneath the Demo.

Demo:

In order to get the same type of information as presented in the Demo, copy the URL provided below into your application or browser.

2.1.1 cURL

```
$ curl 'https://api.userfeeds.io/ranking/links;asset=ethereum'
```

2.1.2 In JavaScript (browser)

```
<!DOCTYPE html>
<html>
<head>
  <title>Simple ranking for Ethereum:</title>
</head>
<body>
  <script>
    fetch('https://api.userfeeds.io/ranking/links;asset=ethereum')
    .then(function(response) {
      return response.json();
    })
```

(continues on next page)

(continued from previous page)

```
    .then(function(ranking) {
      var div = document.getElementById('ranking');

      for (var i = 0; i < ranking.items.length; i++) {
        var item = ranking.items[i];
        div.innerHTML += i.toString() + '.<a href="' + item.target + '">' + item.
↪target + '</a><br/>';
        div.innerHTML += 'Score: ' + item.score + '<br/><br/>';
      }
    });
  </script>
  <h1>Simple ranking for Ethereum</h1>
  <div id="ranking"></div>
</body>
</html>
```

2.1.3 In JavaScript (node.js)

```
const https = require('https');

const options = {
  hostname: 'api.userfeeds.io',
  port: 443,
  path: '/ranking/links;asset=ethereum',
  method: 'GET'
};

console.log('Simple ranking for Ethereum:\n')

const req = https.request(options, (res) => {
  let data = "";

  res.setEncoding('utf8');
  res.on('data', (chunk) => {
    data += chunk;
  });
  res.on('end', () => {
    let ranking = JSON.parse(data);
    for (let index in ranking.items) {
      console.log(`${index}. ${ranking.items[index].target}`);
      console.log(`Score: ${ranking.items[index].score}\n`);
    }
  });
});

req.on('error', (e) => {
  console.error(e);
});

req.end();


// Simple ranking for Ethereum:
//
// 0. http://example.com/one
```

(continues on next page)

(continued from previous page)

```
// Score: 123.1234324
//
// 1. http://example.com/two
// Score: 32.234542343
```

2.1.4 In python

```
import requests

RANKING_URL = "https://api.userfeeds.io/ranking/links;asset=ethereum"

response = requests.get(RANKING_URL).json()

print("Simple ranking for Ethereum:\n")

for index, item in enumerate(response['items']):
    print("{0}. {1}".format(index, item["target"]))
    print("Score: {0}".format(item['score']))
    print()


# Simple ranking for Ethereum:
#
# 0. http://example.com/one
# Score: 123.1234324
#
# 1. http://example.com/two
# Score: 32.234542343
```

2.2 How to get all messages from ERC721 token (such as CryptoKitty Captain Barbosa)

Note: It is important to remember that the ERC721 is recognized as one of standards for tokens.

Demo:

In order to get all messages from ERC721 token, copy the URL provided below into your application or browser.

2.2.1 cURL

```
$ curl 'https://api.userfeeds.io/ranking/feed;
↪context=ethereum:0x06012c8cf97bead5deae237070f9587f8e7a266d:134330'
{"items":[{"about":null,"abouted":[],"author":
↪"0x6be450972b30891b16c8588dcbc10c8c2aef04da","context":"ethereum:0x0...
```

2.2.2 In JavaScript (browser)

```
<!DOCTYPE html>
<html>
<head>
  <title>Simple ranking for Ethereum:</title>
</head>
<body>
  <script>
    fetch('https://api.userfeeds.io/ranking/cryptopurr_feed;
↪context=ethereum:0x06012c8cf97bead5deae237070f9587f8e7a266d:134330')
    .then(function(response) {
      return response.json();
    })
    .then(function(ranking) {
      var div = document.getElementById('ranking');

      for (var i = 0; i < ranking.items.length; i++) {
        var item = ranking.items[i];
        div.innerHTML += i.toString() + '. ' + item.target.id + '<br/>';
      }
    });
  </script>
  <h1>Simple ranking for Ethereum</h1>
  <div id="ranking"></div>
</body>
</html>
```

2.2.3 In JavaScript (node.js)

```
const https = require('https');

const options = {
  hostname: 'api.userfeeds.io',
  port: 443,
  path: '/ranking/cryptopurr_feed;
↪context=ethereum:0x06012c8cf97bead5deae237070f9587f8e7a266d:134330',
  method: 'GET'
};

console.log('Simple ranking for Ethereum:\n')

const req = https.request(options, (res) => {
  let data = "";

  res.setEncoding('utf8');
  res.on('data', (chunk) => {
    data += chunk;
  });
  res.on('end', () => {
    let ranking = JSON.parse(data);
    for (let index in ranking.items) {
      console.log(`${index}. ${ranking.items[index].target.id}`);
    }
  });
```

(continues on next page)

(continued from previous page)

```
});

req.on('error', (e) => {
  console.error(e);
});

req.end();


// Simple ranking for Ethereum:
//
// 0. http://example.com/one
// Score: 123.1234324
//
// 1. http://example.com/two
// Score: 32.234542343
```

2.2.4 In python

```
import requests

RANKING_URL = "https://api.userfeeds.io/ranking/cryptopurr_feed;
↪context=ethereum:0x06012c8cf97bead5deae237070f9587f8e7a266d:134330"

response = requests.get(RANKING_URL).json()

print("Simple ranking for Ethereum:\n")

for index, item in enumerate(response['items']):
    print("{0}. {1}".format(index, item["target"]["id"]))
    print()
```

2.3 How to get all bots (ERC721 tokens) owned by given address

Note: It is important to remember that the ERC721 is recognized as one of standards for tokens.

In order to get all CryptoBots owned by a given address, copy the URL provided below into your application or browser.

2.3.1 cURL

```
$ curl 'https://api.userfeeds.io/ranking/tokens;
↪identity=0x6be450972b30891b16c8588dcbc10c8c2aef04da;
↪asset=ethereum:0xf7a6e15dfd5cdd9ef12711bd757a9b6021abf643'
{"items":[{"sequence":5289388,"token":"4085"}]}
```

CHAPTER 3

Data Model - Claims

Claims are the smallest self-contained pieces of information - an arbitrary information package - inside the Userfeeds Platform. They are also the only way of introducing any information to databases which is sent via the so-called ransports and then interpreted by algorithms in order to return the equested information to interfaces.. All claims are signed cryptographically or have a reference to their cryptographic origin which can be a transaction on the blockchain, for example.

3.1 Structure

The typical and basic structure of a *claim* can be presented in the following way:

from who ; **for whom** ; **how much / amount** ; **transaction identifier**.

The format of *claims* is rather isolated from structure of data in a block-chain - and this is why it is in the json format

3.2 Types

There are several types of *claims* available depending on needs:

- *Claim* with a payment value for a defined account - usually used for getting to a higher position in a ranking – the payment value is the score – it is the algorithm which decides if the structure of such *claim* is proper (it is transation with use of token)
- *Claim* serving for connecting sensual or graphic data with some transaction data
- *Claim* serving for limiting the number of users who can apply for transactions by defining parameters for such transactions and users

Claims which do not bear or transfer a value (from the main Ethereum net) can be used on any block-chain as no token for the identification is needed. Therefore a normal *claim* without a value should be rather used outside of the Ethereum main net.

3.2.1 Overview

The *Claim* may look as follows:

```
{
  "context": "CONTEXT",
  "type": ["TYPEA", "TYPEB", "..."],
  "claim": {
    "target": "TARGET",
    "additional": "fields",
    "go": ["here", "..."]
  },
  "credits": [
    {
      "type": "interface",
      "value": "INTERFACE IDENTIFIER"
    }
  ],
  "signature": {
    "type": "TYPE",
    "creator": "CREATOR",
    "signatureValue": "SIGNATURE"
  }
}
```

3.2.2 `Context`

Note: Populating this field is optional.

The **Context** field is used to denote a *destination* of a given claim. It can be interpreted as a name of any topic or thing about which people might want to share information. Usually it will have something to do with a blockchain space (but it doesn't have to). For example if you would like to share some information to all people interested in *Ethereum* blockchain you will send a claim with `ethereum` as the context.

Other examples of contexts may be:

- ethereum
- ethereumclassic
- bitcoin
- ethereum:0x4564567890abcdef....abc
- ethereumclassic:0x4564567890abcdef....abc
- myspecialcontext
- companyx
- companyx:departmenty

`ethereum:0x123....456` context identifiers are interpreted as *object 0x123....456 on ethereum* and usually will be used to share information about contracts / addresses on a given blockchain.

Special contexts such as starting with *userfeeds:* are *technical* and have a special meaning in the Userfeeds Platform, eg. *userfeeds:pairing* will create a special *PAIRED* relationship allowing one to connect their crypto-currency holdings with an additional public key which will only be used to sign claims.

3.2.3 `Type`

Note: Populating this field is optional.

The Type describes an additional data present in a *claim* object.

An example can be the *labels* type. The object of *claim* of that type needs to have a *labels* key with an array of values.

The description of all supported types can be found at *Types*

3.2.4 `claim`

This key is used to store user provided information and it is mandatory for all claims. `claim` object will always have a `target` key and additional fields depending on the `type` array.

`target`

`target` value identifies a target object which the user wants to share or tag with additional information.

Examples of proper `target` values are:

- http://some.url/path/
- text:base64:base64encodedtext
- ipfs:somehash
- ipdb:somehash
- claim:claimsignaturehash
- mediachain:somehash
- isbn:0451450523
- isan:0000-0000-9E59-0000-O-0000-0000-2
- btih:c12fe1c06bba254a9dc9f519b335aa7c1367a88a
- ethereum:0x4564567890abcdef...abc
- bitcoin:0x4afebcdef...123

Additional fields

That depends on the `type` array additional keys which might be present in `claim` object. See the *Types* for supported types.

3.2.5 `Signature`

This field is generated in order to denote the ownership of claim and the content can be a cryptographic signature or a pointer to the cryptographically secured origin of claim.

Type

Describes what type of signature we are dealing with. It could be `ecdsa.secp256r1` for the elliptic curve signature or `ethereum.transaction` for the claim origination from the Ethereum blockchain.

Supported Signature Types

`ecdsa.prime192v1` In python: ecdsa.NIST192p

`ecdsa.secp256r1` / `ecdsa.prime256v1` In python: ecdsa.NIST256p

`ecdsa.secp224r1` In python: ecdsa.NIST224p

`ecdsa.secp384r1` In python: ecdsa.NIST384p

`ecdsa.secp521r1` In python: ecdsa.NIST521p

`ecdsa.secp256k1` In python: ecdsa.SECP256k1

`ethereum.transaction`

> Claims posted on ethereum blockchain will be verified by comparing the blockchain content with the claim content.

Creator

This identifies a public key or an address which signed the claim.

Format: identifier

- hex:04861127b14bf0036e...ef7127b114988057
- rinkeby:0x1234567890abcdef...1456
- ethereum:0x1235...145
- bitcoin:0x123456...1234

Signature Value

This key holds a raw signature value as produced by the signing algorithm or it can be a transaction hash or any valid identifier of some externally verifiable origin of claim.

3.3 Types

3.3.1 Basic

Basic claim:

```
{
  "claim": {
    "target": "http://some.url/path/"
  },
  "context": "ethereum:0x4564567890abcdef....abc",
  "signature": {
```

(continues on next page)

(continued from previous page)

```
    "creator":
↪"94d1aa6655d931294d524cf52b0df866976f89774bac38a730cf20e2d51dd24d34efc2bbb4d5bba91a7a6582511491dde1
↪",
    "signatureValue":
↪"304402203dac2176721d7e05cd8c580a27a504b64b0a8ee171b18a07630201cbed979ac7022013faf8735f90b957ca4656
↪",
    "type": "ecdsa.prime256v1"
  }
}
```

With the acknowledgment of the interface from which the claim was created:

```
{
  "claim": {
    "target": "http://some.url/path/"
  },
  "context": "ethereum:0x4564567890abcdef....abc",
  "credits": [
    {
      "type": "interface",
      "value": "http://blog.example.com/path/"
    }
  ],
  "signature": {
    "creator":
↪"0df1d4915347bcae90a0696c9efd6300e33b610d31130c3049d329fa61af138de7a7ee55f99057fd8d39c4664be9f1c343
↪",
    "signatureValue":
↪"304502206c243684007c9e412612b5d1a371b20eb146652e4b149bb1fc0e6da437e7f728022100b8c77983949feac478de
↪",
    "type": "ecdsa.prime256v1"
  }
}
```

3.3.2 Link

Additional keys:

- `title` **string**
- `summary` **string**

```
{
  "claim": {
    "summary": "summary",
    "target": "http://some.url/path/",
    "title": "title"
  },
  "context": "rinkeby:0xfe5da6ae3f65e7d5ce29121a9f5872ffd83b45e6",
  "credits": [
    {
      "type": "interface",
      "value": "http://blog.example.com/path/"
    }
  ],
  "signature": {
```

(continues on next page)

(continued from previous page)


```
    "creator":
↪"ffc8c2f39e8a302bf9ca37b06fa9014f0cd3c85900c3d8b771f31a91ce33c050948faedad14d73a4f6c41f5937c3a010b5
↪",
    "signatureValue":
↪"304602210093c55c30be1868de005def995aefb391d7ff20f235457b37a01e6921427701f80221008e5fc2afc2c912466
↪",
    "type": "ecdsa.prime256v1"
  },
  "type": [
    "link"
  ]
}
```


3.3.3 Labels

Additional keys:

- `labels` **array** of **string**


```
{
  "claim": {
    "labels": [
      "Good",
      "Book",
      "Cats"
    ],
    "target": "http://some.url/path/"
  },
  "context": "rinkeby:0xfe5da6ae3f65e7d5ce29121a9f5872ffd83b45e6",
  "signature": {
    "creator":
↪"de9965ce03cf6f960a7efe423633409a0052ad8f9f2100e27026ad94551d4d69058c0a263dbd0cacf999ca3e97ddcc0afa
↪",
    "signatureValue":
↪"3044022069457927f1fc06b26467a7cc93c99085efea4d8811c6979ffab9ba2196be5ad702201587d3f88d2e9058d6ce48
↪",
    "type": "ecdsa.prime256v1"
  },
  "type": [
    "labels"
  ]
}
```


3.4 Value Transfer

Along with claims the Userfeeds Database contains normalized data about transfer of assets (tokens and others)

If the transfer of assets was accompanying a claim, they will be connected with the *TRANSFER* relation.

CHAPTER 4

Contracts

Contracts can be decsribed as notaries or agents that mediate between *claims* and blockchains. Three types of contracts can be distinguished:

- Notary only - for non-value transations and *claims*
- Allowing to pass a value (money) to a given address (acting as a proxy contract)
- For the token ERC20 only (which serve as money-related transations) - an example here can be a contract designed for managing split payments

CHAPTER 5

Web Components

5.1 Button

…

CHAPTER 6

API Reference

The API root is available at https://api.userfeeds.io/

6.1 Retrieving Data

read-only

The available algorithms are described in the *Algorithms* section.

```
Schema:

$ curl https://api.userfeeds.io/ranking/algorithm1Name;param1=value1;param2=value2.../
↪algorithm2Name...

An example:

$ curl https://api.userfeeds.io/ranking/links;asset=ethereum
```

GET /ranking/algorithm1Name;param1=value1;param2=value2.../algorithm2Name...

An example request:

An example response:

OR

```
Schema:

$ curl -X POST -v -d '{"flow":[{"algorithm":"algorithm1name","params":{"param1":
↪"value1",...},...}]}' -H 'Content-Type: application/json' 'https://api.userfeeds.io/
↪ranking/'

An example:

$ curl -X POST -v -d '{"flow":[{"algorithm":"links","params":{"asset":"ethereum"}}]}'␣
↪-H 'Content-Type: application/json' 'https://api.userfeeds.io/ranking/'
```

(continues on next page)

(continued from previous page)

POST /ranking/

An example request:

An example response:

CHAPTER 7

Algorithms

The Userfeeds Platform allows running custom algorithms on the top of data gathered from all supported sources such as Ethereum blockchain (mainnet, ropsten, rinkeby, kovan)

Algorithms can be referenced by their identifier which depends on how the author has decided to share them.

Note: The support is currently limited to built-in algorithms created by Userfeeds only. We will open custom algorithms for external developers in the near future.

7.1 Available built-in algorithms

CHAPTER 8

Transports

Transports are manners of passing claims to blockchains. They can simply be seen as transport layers. Transports can also be seen as intermediaries between claims and blockchains.

8.1 HTTP

You can send signed *claims* through the HTTP Gateway of the Userfeeds Platform.

Properties of transports are as follows:

In-transport secrecy The *claim* cannot be sniffed on transport thanks to a HTTPS connection and will only be available to the outside world after it is incorporated into the Userfeeds Platform through APIs rankings and database dumps.

Independent distribution The *claim* exists only inside the Userfeeds Platform database and will be distributed with Userfeeds Platform database dumps.

An example of posting a *claim* directly to the Userfeeds Platform through the HTTP transport:

```
$ curl \--=der43
  -X POST https://api.userfeeds.io/storage/ \
  -H "Content-Type: application/json" \
  -H "Authorization: 59049c8fdfed920001508e2a03414df648e34ea665f544a17d5c113b" \
  -d '{"claim":{"target":"http://some.url/path/"},"context":
↪"ethereum:0x4564567890abcdef....abc","signature":{"creator":
↪"82fb68fc14719b94b36e99e588c9988458ca187d4791463164285bb064458232c4bb5bb638158096f4ed957e275e2e157
↪","signatureValue":
↪"c675d123fb1e99ffe59e7626c655806ebe47ec1ca4100b953029731159c2e14f4461e2ebf7f43a071b847b57cbcbd66bd
↪","type":"ecdsa.prime256v1"}}'
```

8.2 Ethereum Transaction

You can send a *claim* through an Ethereum Blockchain transaction.

Note: The Userfeeds Platform does not monitor every transaction for potential claims yet.

In order to send a *claim* through a transaction you need to call special contracts which are monitored by the Userfeeds Platform.

Properties:

In-transport secrecy The *claim* is available from the moment it is distributed through the Ethereum network and can be sniffed before it reaches its desired ranking.

Independent distribution The *claim* will be a part of the Etherum Blockchain and will be available from all copies of the blockchain.

8.2.1 Contracts

With a value transfer:

Code

```
pragma solidity ^0.4.11;

contract Userfeeds {

    event Claim(address sender, address userfeed, string data);

    function post(address userfeed, string data) payable {
        userfeed.transfer(msg.value);
        Claim(ms4g.sender, userfeed, data);
    }
}
```

ABI

```
[
  {
    "constant":false,
    "inputs":[
      { "name":"userfeed", "type":"address" },
      { "name":"data", "type":"string" }
    ],
    "name":"post",
    "outputs":[],
    "payable":true,
    "type":"function"
  },
  {
    "anonymous":false,
    "inputs":[
      { "indexed":false, "name":"sender", "type":"address" },
      { "indexed":false, "name":"userfeed", "type":"address" },
      { "indexed":false, "name":"data", "type":"string" }
    ],
    "name":"Claim",
    "type":"event"
  }
]
```

Addresses

- For Mainnet: ...
- For Rinkeby: 0x0a48ac8263d9d79768d10cf9d7e82a19c49f0002
- For Ropsten: 0xa845c686a696c3d33988917c387d8ab939c66226
- For Kovan: ...

Without a value transfer:

Code

```
pragma solidity ^0.4.11;
contract Userfeeds {

    event Claim(address sender, string data);

    function post(string data) {
        Claim(msg.sender, data);
    }
}
```

ABI

```
[
  {
    "constant":false,
    "inputs":[
      { "name":"data", "type":"string" }
    ],
    "name":"post",
    "outputs":[],
    "payable":false,
    "type":"function"
  },
  {
    "anonymous":false,
    "inputs":[
      { "indexed":false, "name":"sender", "type":"address" },
      { "indexed":false, "name":"data", "type":"string" }
    ],
    "name":"Claim",
    "type":"event"
  }
]
```

Addresses

- For Mainnet: ...
- For Rinkeby: 0x09dcdf34e0c28b106fdfe51009cb71ae92bf8bbc
- For Ropsten: 0x5c3fe6b94b57c1e294000403340f12f083e71b83
- For Kovan: ...

8.3 Whisper Protocol

TODO: Direct message TODO: Broadcast

8.4 IPDB (BigchainDB)

TODO

CHAPTER 9

Indexes and tables

- genindex
- modindex
- search

HTTP Routing Table

/ranking