Cosmic Index of Public Resources

Juan Barriteau

Last update: 15/11/2023, 09:33:20 UTC-0400

Index

List of Figures

Cipr: Semi-informal Specification

> **Note**: This is a first iteration of this document, just an early exposition of the idea, room for improvement is huge, suggestions and corrections are more than welcome.

The *Cosmic*[1] *Index of Public Resources*, or *Cipr*[2], is a decentralized, distributed, independent, public, universal, dynamic and searcheable directory of websites and other reachable-by-DNS-resolution resources in the Internet.

The Cipr shares some characteristics with conventional search engines and with classic web directories, but adding entries to the Cipr doesn't requires crawling the web, as most search engines do; neither requires the approval of curators or editors, as most directories do.

With the Cipr every content publisher—every domain holder—*owns* their entries in the index, meaning, they create, modify and destroy them at will.

[1]Because Martians and Belters are welcome.

[2]Pronounced like kai-per? cy-per? see-per? kee-per? I use the last one, but I'm not good at English.

Factors that determine ranking position of search results can't be obscured in the Cipr, they are standardized, consistent, public and defined by universal consensus.

SEO in the Cipr is just about using the correct titles, descriptions, keywords, alerting and localization data, nothing else.

Censoring/banning/blocking/filtering a Cipr indexed resource is only possible through DNS censoring/banning/blocking/filtering.

The global availability of every insertion, modification or deletion to the Cipr is expected to take only a few minutes, when no less.

Having a website or any other resource effectively indexed in the Cipr simply implies:

- Deploying a specific daemon.
- Filling the resource entry (title, description, keywords...).
- Adding a specific `TXT` record in the corresponding DNS Zone namespace.

The mentioned *specific daemon* is a ciprnode instance, and the *specific TXT record* is a simply hashed string provided by the deployed ciprnode. This process is detailed in next sections, but it's worth to mention that, due to the simplicity of a ciprnode, the possible surge of free or low cost *ciprnode hosting services* is expected to make this dead simple.

The environment that makes possible the Cipr's existence is the Ciprsys.

Ciprsys

Ciprsys is the set of software components, network elements, protocols, services, principles and constrains that guarantees the completeness, integrity, availability, responsiveness, accuracy, reliability and up-to-dateness of the the Cipr. Those components are:

- **Domain Name System**: the existing Internet's naming system
- **Ciprnet**: network created by the acting agents in the Ciprsys
- **Ciprnode**: each Ciprsys daemon
    - **Ciprdup**: copy of the Cipr in every ciprnode
        - **Ciprule**: conventions for FTS[3] in the ciprdup
    - **Localindx**: indexed content of the ciprnode's domain
    - **Ciprbot**: communications agent in every ciprnode
        - **CiprAPI**: the API exposed by the every ciprbot
        - **Ciprpulse**: set of reliability and up-to-dateness checks
    - **Ciprface**: Cipr search web client

[3]Full Text Search.

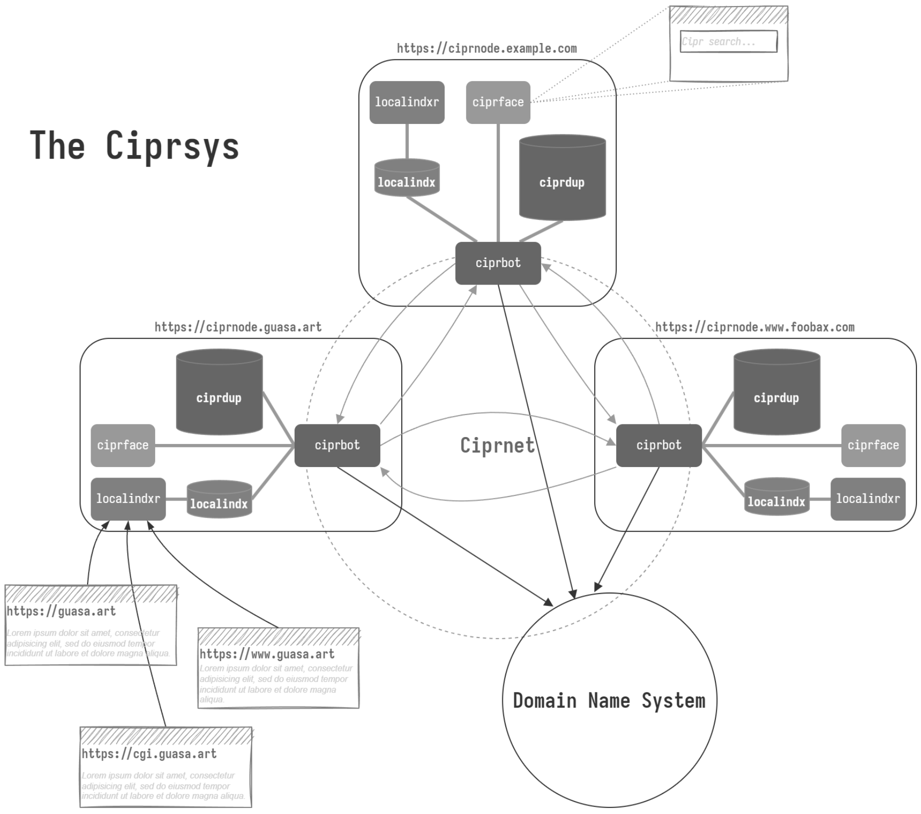

Figure 1: The Ciprsys

Every resource effectively indexed in the Cipr is referred in this document as a *cipred resource*.

The FQDN of a cipred resource will be mentioned as the cFQDN, the same as a `sldl.tldl`.

Domain Name System

The DNS is the old, trustable, ubiquitous hierarchical and decentralized naming system used to identify resources in the Internet; Ciprsys uses it to:

1. **Verify cipred resources existence and validity** by finding existing entries when looking up for a resource.
2. **Verify Cipr entries correctness** by matching hashes of Cipr entries with existing TXT records.

Extending the verification tasks to any known DNS Root Zone alternative[4] is technically possible.

Ciprnet

The Ciprnet is the network created by the acting agents in the Ciprsys. There are three types of message exchanges in the Ciprnet:

1. Internal communication between the daemons of the network.
2. Communication between the daemons of the network and the DNS.
3. Communication between external clients and the daemons of the network.

[4]Handshake, OpenNIC, Namecoin…

Ciprnode

Most functions of the Ciprsys relies on its ciprnodes. A ciprnode is a daemon with the main function of holding a copy of the Cipr and keep it synchronized with all other copies in the rest of the ciprnodes.

Second function of a ciprnode is to serve as an entry point for the search queries to the Cipr, this is done by exposing a web service to attend non-human request, and a web form for human users.

Additionally, each ciprnode must index the full content of the resource it belongs to, and search queries to this internal/local index must be possible through the mentioned human and non-human interfaces.

Every ciprnode must be accesible in a FQDN following this pattern:

`https://ciprnode.sldl.tldl`

Where `tldl` is a *top level domain* (TLD) label, `sldl` is a *second level domain* (SLD) label, the literal *ciprnode* **must** be the *third level domain* (3LD) label and no further subdomains can exist under it, for example:

`https://ciprnode.cipr.info`

Note that for some *country code top-level domains* (ccTLD) is limited or forbidden the registration of *second level domains*, this means that resources like `bbc.co.uk`, `up.edu.br` or `ivic.gob.ve` can't be indexed in the Cipr[5].

[5]This is because allowing ciprnodes under the 3LD will permit the incorporation of infinite ciprnodes under one FQDN.

Ciprdup

A ciprdup is the working copy of the Cipr in every ciprnode, it's probably—but not mandatorily—a table or a group of tables in a DBMS. The fields of the ciprdup are the same fields of the Cipr plus one: `sldl`, `tldl`, `title`, `description`, `keywords`, `geo`, `ofle` and `ban`.

Ciprdup fields The ciprdup's fields are essentially the Cipr's fields, let's digg in one each of them.

sldl *Second level domain* label of the resource in the Domain Name System.

- **Constrains:**
  - Min length: 1 character.
  - Max length: 64 characters.
  - Allowed values: all the Unicode charset except the *full stop* character (.).

tldl *Top level domain* label of the resource in the Domain Name System.

- **Constrains:**
  - Min length: 1 character.
  - Max length: 64 characters.
  - Allowed values: all the Unicode charset except the *full stop* character (.).

title The indexed resource's title, this field is to be included in FTS queries.

- **Constrains**:
    - Min length: 0 characters.
    - Max length: 64 characters.
    - Allowed values: all the Unicode charset and nothing.
    - Default: empty.

description The resource's description, this field is to be included in FTS queries.

- **Constrains**:
- Min length: 0 characters.
- Max length: 256 characters.
- Allowed values: all the Unicode charset and nothing.
- Default: empty.

keywords The resource's keywords, this field is to be included in FTS queries.

- **Constrains**:
    - Min length: 0 characters.
    - Max length: 512 characters.
    - Allowed values: all the Unicode charset and nothing.
    - Default: empty.

geo Approximate geographic coordinates of the resource, this is an optional field, when used coordinates are stored with very low precision.

- **Constrains**:
  - Min length: 0 character.
  - Max length: 14 characters.
  - Allowed values: nothing or a comma separated longitude and latitud pair with a max of two decimal places each.
  - Default: empty.

ofle Offensiveness level, a subjective indicator of how offensive the resource content could be from its publisher's point of view. Possible values are:

NOC (empty): *Non Offensive Content*, indicates the content is not offensive to any social group in the whole world, including but not restricted to: specific regions inhabitants, ethnic groups, cultural groups, any nation members, guilds members, collectives members, tribes members, age groups members, genre groups members, religious groups members, or members of any other form of social aggrupation.

LOC (1): *Localized Offensive Content*, indicates the content could be offensive to one or more social groups, but not to all of them.

UOC (2): *Universally Offensive Content*, used when the publisher considers the offensiveness of the content is transversal to most social groups in the world.

Offensiveness for individuals is not to be considered, but it's up to the publishers assuming specific individuals as groups.

It is suggested to provide extra information in the description field to clarify why the resource is considered LOC or UOC when this is the case.

- **Constrains**:
  - Min length: 0 characters.
  - Max length: 1 character.
  - Allowed values: `1`, `2` and empty.
  - Default: empty.

ban Auxiliary column where a ciprbot stores its own assessment of a resource trustability. An empty or false value means the resource is trusted, `true` indicates distrust.

This column is for the internal use of each ciprbot and mustn't be shared between nodes.

It's expected that a ciprbot ignores requests coming from banned peers and avoid querying them, more details about this in the section dedicated to the ciprpulse.

- **Constrains**:
  - Min length: 0 characters.
  - Max length: 1 character.
  - Allowed values: `true|false`.
  - Default: empty.

General constrains `sldl` and `tldl` are to be considered a *primary key* in the ciprdup, meaning, there can't be two entries with the same combination of them.

Example

Table 1: Example representation of the Cipr

sldl	tldl	title	description	keywords	geo	ofle	ban
`example`	`com`	`Example Domain`	`For examples`	`rat pote table`			`true`
`foobar`	`org`	`The Foobar Zone`	`Foobar`	`late chupe ola`	`-90.12,-180.12`		
`elcoco`	`buh`	`Offense For All`	`Fully offensive`	`bit cigar tool`		`2`	
`offense`	`com`	`Offense For Some`	`A bit offensive`	`truck ala wing`		`1`	
`pali`	`to`	`Little Stick`	`Stick dedicated`	`polo hilo star`			`true`
`cipr`	`info`	`The Cipr Spec`	`Cipr spec`	`pose wind pork`	`1,1`		

Ciprule The Ciprule is a universal convention about how optimization, filtering and ranking are to be applied when searching in the ciprdups.

Ciprule defines the criteria to rank results, how to strip if stripping is allowed, how to tokenize if tokenizing is a thing, if lowercasing or not, if stop words are removed or not and which ones, which are synonyms if they are to be considered, if stemming or not, if using fuzziness or not, how to treat ORs, ANDs and NORs, exact phrases and so on.

This is not something to be defined at this stage of this document, so, more to come.

Localindx

The localindx is a full content index of a individual cipred resource. The creation of the localindx is the exclusive responsibility of each publisher, how (crawling? at building?) and when (daily? weekly? monthly?) depends on them; but, no matter how/when it is generated, the localindx must be searchable through the CiprAPI.

The use of a localindx isn't mandatory, but having it is extremely convenient for the publisher; this is an optional but desirable component.

Ciprbot

A ciprbot is the residing autonomous agent in every ciprnode, it executes some tasks on demand and some tasks following a predefined schedule; basically, they keep all ciprdups in sync while serving as entry points for Cipr searches.

CiprAPI CiprAPI is the program interface used to keep the flow of search queries and the integrity and synchronization of all ciprnodes, this is, guaranteeing that every one of them effectively keeps an updated, trustable and queryable copy of the Cipr.

Each ciprbot exposes *at least* a REST-compliant interface exchanging *at least* a set of JSON:API formatted messages. The minimal essential endpoints in this interface are:

Summary of the CiprAPI's endpoints

```
SEARCH /
SEARCH /cFQDN/

PUT /
PUT /cFQDN/
PUT /cFQDN/title/
PUT /cFQDN/description/
PUT /cFQDN/keywords/
PUT /cFQDN/geo/
PUT /cFQDN/ofle/
PUT /cFQDN/ban/

DELETE /
DELETE /cFQDN/
DELETE /cFQDN/title/
DELETE /cFQDN/description/
DELETE /cFQDN/keywords/
DELETE /cFQDN/geo/
DELETE /cFQDN/ofle/
DELETE /cFQDN/ban/

GET /
GET /cFQDN/
GET /cFQDN/title/
GET /cFQDN/description/
GET /cFQDN/keywords/
GET /cFQDN/geo/
GET /cFQDN/ofle/
GET /cFQDN/ban/

OPTIONS *
OPTIONS /
OPTIONS /cFQDN/
OPTIONS /cFQDN/title/
OPTIONS /cFQDN/description/
OPTIONS /cFQDN/keywords/
OPTIONS /cFQDN/geo/
OPTIONS /cFQDN/ofle/
OPTIONS /cFQDN/ban/
```

The `SEARCH` methods can receive the `pages[num]` and `pages[size]` query parameters, being `num` an array of integers indicating which page numbers are expected per query, and `size` an array of integers indicating the expected number of entries per page on each query. For example:

```
SEARCH /cFQDN/?pages[num]=[1,1]&pages[size]=[10,20]
```

The same parameters can be sent included in the body request and, in case of conflict, the body has precedence.

The `GET` methods can receive the `page[num]` and `page[size]` query parameters, being `num` an integer indicating which page number are expected, and `size` an integer indicating the expected number of entries per page. For example:

```
GET /cFQDN/keywords/?page[num]=1&page[size]=20
```

In the body of every `OPTIONS` response are described the requests and responses schemas used with each method.

Basic characteristics of the CiprAPI Implementations of the CiprAPI must have full Unicode support and must be, at least, JSON:API compliant, thus, all HTTP requests with body have to validate with this standard, the same applies for all responses with body. When using this standard, the `Content-Type` and `Accept` headers must have `application/vnd.api+json; charset=utf-8` as their media type.

Unrecognized media types must treated as `text/html` and redirected to the ciprface of the deployed node, the same applies if the `Accept` header isn't present in the petition.

The CiprAPI must adhere at least *and strictly* to the whole REST architectural constraints, that paying particular attention to the *uniform interface constraint* and the use of *hypermedia as the engine of application state*; this is *vital* in order to guarantee that, after any update, change or alteration of the API, all the automaton agents in the Ciprsys stay able to work after recognizing every change and adapting to it without human intervention.

The mandatory RESTfulness of the CiprAPI doesn't means that implementations adding different architectural styles are prohibited, as far as the required level of automaton's autonomy is provided, anything else is perfectly acceptable, and even desirable.

The same principle applies to the use of JSON:API and the HTTP protocol itself, they all are mandatory as defaults, but developers finding better or more efficient means to achieve the expected functionalities are encouraged to improve the Ciprsys with their ideas.

Non SSL/TLS requests to the CiprAPI must be rejected, always and under any circumstance.

Ciprpulse The Ciprpulse is a set of automated reliability and up-to-dateness checks in the Cipr, it consist of, at least, eight checks:

- **Reliability checks**:
    1. `SEARCH` rebroadcasting
    2. `SEARCH` broadcasting
- **Up-to-dateness checks**:
    3. `PUT` rebroadcasting
    4. `PUT` broadcasting
    5. `DELETE` rebroadcasting
    6. `DELETE` broadcasting
    7. DNS entries check
    8. Banned entries check

Rebroadcasting: refers to the act of copying an incoming CiprAPI request and, simultaneously, send identical requests to one or more randomly chosen cFQDNs.

Broadcasting: refers to the act of simultaneously sending an identical request to one or more randomly chosen cFQDNs; in this case, the request is created by the ciprbot itself, isn't the copy of an incoming one.

Rebroadcastings are triggered by the reception of specific requests, *broadcastings* are triggered by schedule.

Additional checks could be added, this is just a mandatory minimal set, if it's considered that other checks will improve the reliability and up-to-dateness of a implementation with no detriment of any other aspect of the Ciprsys, their addition is perfectly acceptable.

A ciprbot implementation must use, at least, the implementations of the following internal functions[6]:

```
/**
 *
 * Retrieve n randomly selected
 * cFQDNs from the ciprdup.
 *
 * If n is null, the function randomly
 * chooses how many cFQDNs return.
 *
 * @param {?number} n - number of expected cFQDNs
 * @returns {string[]} - array of cFQDNs
 */
const randomEntriesSet = (n) => { /* ... */ };

/**
 *
 * Retrieve n randomly selected
 * banned cFQDNs from the ciprdup.
```

[6]This is language independent pseudocode, any similarity with ECMAScript and JSDoc syntaxes is coincidental.

```
 *
 * If n is null, the function randomly
 * chooses how many cFQDNs return.
 *
 * @param {?number} n - number of expected banned cFQDNs
 * @returns {string[]} - array of banned cFQDNs
 */
const randomBannedEntriesSet = (n) => { /* ... */ };

/**
 *
 * Retrieve a random period of time.
 *
 * @returns {integer} - time period in miliseconds
 */
const randomPeriod = () => { /* ... */ };

/**
 *
 * Generates a fake query.
 *
 * Randomized aspects must include all
 * possible variables: keywords, page
 * number, page size, geo, ofle...
 *
 * @returns {Object[]} - array of objects with a fake query
 */
const fakeQuery = () => { /* ... */ };
```

```
/**
 *
 * Send a set of queries to the local ciprdup.
 *
 * @param {Object[]} q - array of objects of queries
 * @returns {Object[]} - array of objects with search results
 */
const ciprdupSearch = (q) ⇒ { /* ... */ };

/**
 *
 * Send a set of queries to the localindx.
 *
 * @param {Object[]} q - array of objects of queries
 * @returns {Object[]} - array of objects with search results
 */
const localindxSearch = (q) ⇒ { /* ... */ };

/**
 *
 * Query for especific entries in the local ciprdup.
 *
 * @param {string[]} e - array of cFQDNs
 * @returns {Object[]} - array of objects with the requested data
 */
const localRequest = (e) ⇒ { /* ... */ };

/**
 *
 * Insert/update to the local ciprdup.
 *
 * @param {Object[]} e - array of entries to be processed
 * @returns {boolean[]} - array of booleans indicating success or not
 */
const localChange = (e) ⇒ { /* ... */ };

/**
 *
 * Delete entry or entries in
 * the local ciprdup.
 *
 * @param {Object[]} e - array of entries to be processed
 * @returns {boolean[]} - array of booleans indicating success or not
 */
const localRemoval = (e) ⇒ { /* ... */ };

/**
 *
 * Compare all provided strings
 * and returns the matching factor,
 * being it 1 if all strings match,
 * 0 if none match, 0.5 if half of
 * them match, and so on.
 *
 * @param {string[]} s - array of strings
 * @returns {number} - matching factor
 */
const comparison = (s) ⇒ { /* ... */ };
```

```
/**
 *
 * Returns the total number of
 * entries in the ciprdup.
 *
 * @returns {integer} - total entries
 */
const entriesCount = () => { /* ... */ };

/**
 *
 * Given a list cFQDNs with their hashes, the Domain Name
 * System is queried to verify if the provided hashes are
 * matches to their correspondants in the Cipr's TXT record
 *
 * @param {Object[]} d - object of cFQDNs with their hashes
 * @returns {boolean[]} - array of booleans
 */
const lookup = (d) => { /* ... */ };

/**
 *
 * Non-cryptographic hash function
 *
 * @param {string} s - string to be hashed
 * @returns {string} - hash
 */
const hash = (s) => { /* ... */ };
```

1. SEARCH rebroadcasting After receiving a `SEARCH` request with a `q` query, a ciprbot must:

1.1. Send the same `SEARCH` request looking for the **SAME PAGE** to a `randomEntriesSet(n)`[7] of cFQDNs.

1.2. Send the same `SEARCH` request looking for the **NEXT PAGE** to a `randomEntriesSet(m)`[8] of cFQDNs.

— For any response to the sent `SEARCH` requests that is erroneous (any `5XX Server Error` HTTP Status Code or timeout):

A `PUT /cFQDN/ban/` with `ban=true` must be sent to a `randomEntriesSet(p)`[9] of cFQDNs, for example:

```
PUT /cFQDN/ban/ HTTP/1.1
Host: ciprnode.sldl.tldl
Content-Type: application/vnd.api+json; charset=utf-8

{
  "data": [
    {
      "ban": true
    }
  ]
}
```

[7] Initial suggested value for `n` is `1`, but a specific equation to get this value must be created.

[8] Initial suggested value for `m` is `4`, but a specific equation to get this value must be created.

[9] Initial suggested value for `p` is the 2% of `entriesCount()`, but a specific equation to get this value must be created.

The intention with the bans broadcasting is to notify in the Ciprnet the existence of faulty ciprnodes. Banning a cFQDN means that requests from it must be responded with `200 OK` HTTP status without taking any other action, and requests to it must be avoided.

— **For any response to the sent `SEARCH` request that has any `4XX Client Error` HTTP Status Code**:

The needed `OPTIONS` request must be sent to the failing ciprbot in order to diagnose what's happening, if it's determined that a local problem exists, corrections must be made and the original requests must be resent, if the diagnostic determines it's really a problem with the requested node, a `PUT /cFQDN/ban/` with `ban=true` must be sent to a `randomEntriesSet(p)` of cFQDNs.

— **For any response to the sent `SEARCH` request that has any `2XX Sucess` HTTP Status Code**:

If it was a `1.1.` request, a `comparison()` must be made between `ciprdupSearch(q)` and every query results coming from `randomEntriesSet(n)`, it is expected all of them to be identical, if not, a `PUT` with `ban=true` for the failing ones must be sent to a `randomEntriesSet(p)`, being *the failing ones* those different to `ciprdupSearch(q)`.

If it was a `1.2.` request, a `comparison()` must be made between all the query results coming from `randomEntriesSet(m)`, it is expected all of them to be identical, if not, a `PUT` with `ban=true` must be sent to a `randomEntriesSet(p)` for the failing ones ; being *the failing ones* the fewest that differ from the majority.

The assumed correct **NEXT PAGE** is to be temporarily stored and used if asked by the client.

2. SEARCH broadcasting Here happens *almost* the same as in `1.1.`, only difference is that here the ciprbot sends a `SEARCH` request with a `fakeQuery()` to a `randomEntriesSet(n)` every `randomPeriod()` and not because of a received request, it's just an scheduled task, the rest of the steps here are exactly the same ones as in `1.1.`.

3. PUT rebroadcasting After receiving a `PUT` request with the `e` entries, a ciprbot must `lookup(d)` to verify the correctness of the received update/insert, if it's verified, the ciprbot must do a `localChange(e)` and then send the same `PUT` request to a different `randomEntriesSet(n)` of cFQDNs every `randomPeriod()` **until receiving again the exact same request.**

When the `PUT` is a ban request, a `ciprdupSearch(fakeQuery())` must ne compared with the result of a `SEARCH` of the same `fakeQuery()` sent to the suspicious cFQDN, if they are equal, the ban request is ignored, otherwise, the rebroadcasting of the request has to be made.

— **For any response to the sent `PUT` requests that is erroneous (any `5XX Server Error` HTTP Status Code or timeout):**

A `PUT /cFQDN/ban/` with `ban=true` must be sent to a different `randomEntriesSet(n)` of cFQDNs every `randomPeriod()` until receiving again the exact same request.

— **For any response to the sent `PUT` request that has any `4XX Client Error` HTTP Status Code:**

The needed `OPTIONS` request must be sent to the failing ciprbot in order to diagnose what's happening, if it's really a client problem, local corrections must be made and requests must be resent, if the diagnostic determines is really a problem with the requested node, a `PUT /cFQDN/ban/` with ban set to `true` must be sent to a different `randomEntriesSet(n)` of cFQDNs every `randomPeriod()` until receiving again the exact same request.

— **For any response to the sent `PUT` request that has any `2XX Sucess` HTTP Status Code:**

No further actions are to be made.

4. PUT broadcasting Here the ciprbot does a `localRequest(randomEntriesSet()` and sends a group of `PUT` request to a `randomEntriesSet(n)` every `randomPeriod()` **until receiving again the exact same request** for every cFQDN in this `randomEntriesSet()`.

5. DELETE rebroadcasting Here is done exactly the same as in `3.`, only difference is the use of `DELETE`s instead of `PUT`s.

6. DELETE broadcasting The only situation where a ciprbot start a new `DELETE` to be broadcasted is to be removed itself from the Ciprnet after deleting its `TXT` entry from the DNS.

7. DNS entries check The ciprbot sends a `lookup(randomEntriesSet())` every `randomPeriod()`. A `PUT /cFQDN/ban/` must be for each failed check, for all successful checks any action must be taken.

8. Banned entries check The ciprbot sends a `SEARCH` request with a `fakeQuery()` to a `randomBannedEntriesSet()` every `randomPeriod()`. A `comparison()` must be made between `ciprdupSearch(fakeQuery())` and every query results coming from the `randomBannedEntriesSet()`, a `PUT` with `ban=false` must be sent to a `randomEntriesSet()` for each response identical to `ciprdupSearch(fakeQuery())`. For all failed checks any action must be taken and the existing ban stays.

Incorporation to the Ciprnet

The process to incorporate a ciprnode to the Ciprsys is the same process that allows having an entry in the Cipr. In general terms, the following steps mus be taken to have a working ciprnode, and a valid entry in the Cipr:

1. Ciprnode deployment

This step is accomplished after having a ciprnode installed and able to operate in the Ciprnet as `https://ciprnode.sldl.tldl`. This is not about it to be operating as an effective node of the network yet, it's the sole fact of it being ready to receive request to its API, but still being an unknown agent in the Ciprnet.

2. Initial configuration

Each ciprnode must be specifically configured before its incorporation to the Ciprnet. A least, the following parameters must be provided to be stored in a configuration file, a database or something similar:

```
bootstrap_node="a trusted and already incorporated ciprnode URI"
thirdld="subdomain labels"
sldl="second level domain label"
tldl="top level domain level"
title="title of the resource"
description="description of the resource"
keywords="keywords of the resource"
geo="coordinates of the resource"
ofle="level of offesivness"
```

This is a minimal, probably more configuration parameters will be needed, like limits and ranges for the use of the internal functions, timeouts for the different types of requests and so on.

3. Ciprdup population

Once the ciprnode is deployed and has its initial configuration, next action is to populate its ciprdup, the retrieval of entries begins with a `GET /` to the `bootstrap_node` and then it's possible to keep going with the gradually obtained cFQDNs.

Note that, even when a `GET /` is an ask for the whole Cipr, the response is always paginated, then, asking for different pages to different nodes could be wiser than sticking to only one.

Of course, it's perfectly possible to simply copy a whole ciprdup from and existing ciprnode and avoid expending time in the sync process, but in this case is very important to certify the validity of the obtained copy, otherwise, the incorporating ciprnode will be getting banned as soon as it start connecting to the network.

In any case, the last entry to insert in the ciprdup is the corresponding one to the ciprnode itself taking the needed fields from the initial configuration file/db.

4. Hash generation

Having a populated ciprdup, the ciprnode must automatically generate a hash using the existing info in the configuration:

```
const selfGneratedEntryHash = hash(
  `${title}^^${description}^^${keywords}^^${geo}^^${ofle}`
);
```

5. TXT record creation

By manual or automated means, a `TXT` record must be created in the corresponding DNS Zone namespace, something like:

```
Name: ciprnode.sldl.tldl
Record Type: TXT
Value: "entryhash=selfGneratedEntryHash"
TTL: 1800
```

6. Ciprpulse activation

At this point the ciprnode is ready to enter the Ciprnet, for this all the Ciprpulse functions must be activated, plus, it could be a good idea to promote the ciprface usage, search going through it contributes with the sanity of the newly incorporated ciprnode.

Ciprface

A ciprface is a front-end for the human interaction with the Cipr, is the default client of the Ciprsys. At least one ciprface must be available in every ciprnode and it must be accesible from any browser as:

```
https://ciprnode.sldl.tldl
```

Bing this the exact same URL of the same node ciprbot, the ciprnode must be able to discriminate how to process the request based in the information provided by the `Accept` HTTP header; if it's empty, absent or has any of the following media types:

- `text/html`
- `application/xhtml+xml`
- `application/xml;q=0.9`
- `*/*;q=0.8`

The request is for the ciprface, and it it's:

- `application/vnd.api+json; charset=utf-8`

The request is for the ciprbot.

Non-TLS requests to a ciprface must be always ignored, rejected or redirected.

There are no checks to verify the presence of the ciprface in a ciprnode, so they could be absent or deactivated without affecting the reputation of the ciprnode, but having an active ciprface has advantages for a ciprnode: the more search requests it processes, more up-to-date will be its ciprdup.

Some recommendations may be made in relation with the content and design of a ciprface, but every domain holder has the last word about it. That being said, there is a minimal set of features a ciprface must have to be considered compliant with this specification:

- Must offer an input box to enter queries to the Cipr.
- Help and guidance must be given about the Ciprule's available options (case sensitivity, ORs, NORs, ANDs...).
- At least one of the following capabilities must be available:
  - Pagination of results
  - *Load more* results button
  - Infinite scroll for results
- Must have the ability to merge localindxs results with their corresponding ciprdup results (more on this in the next section).
- Must provide the necessary tools to improve a search and recursively optimize it.

Searching in the Cipr

A single request to search the Cipr could contain multiple queries, for example:

```
SEARCH / HTTP/1.1
Host: ciprnode.sldl.tldl
Accept: application/vnd.api+json; charset=utf-8
Content-Type: application/vnd.api+json; charset=utf-8

{
  "data": [
    {
      "srch_id": 1,
      "srch_string": "how to play the drums",
      "srch_options": {
        "page_number": 1,
        "page_size": 3
      }
    },
    {
      "srch_id": 2,
      "srch_string": "cat turpial && toad (conejo || dog) chivo",
      "srch_options": {
        "page_number": 1,
        "page_size": 10
      }
    }
  ]
}
```

A request like this could be made from one ciprnode to another, or from a client to any a ciprnode.

Every Cipr search happens in two phases, the first one returns the *Directory Entries Match Set* (DEMS) and the second returns the *Textual Content Match Set* or TCMS.

Directory Entries Match Set

This is the result of matching the user provided query with the entries in the Cipr itself, it's obtained by full-text-searching the contents of the `title`, `description` and `keywords` fields and applying any provided filtering for the `geo` and `ofle` fields.

The DEMS is basically a paginated list of cipred resources ordered by their relevance to the provided search query, for example:

```
{
  "srch_id": 1,
  "page_num": 1,
  "page_size": 3,
  "total_pages": 234,
  "matches": [
    {
      "rank": 1,
      "domain": "example.com"
    },
    {
      "rank": 2,
      "domain": "example.org"
    },
    {
      "rank": 3,
      "domain": "example.net"
    }
  ]
}
```

Textual Content Matches Set

Immediately after obtaining a DEMS, the client iterates over all its cFQDNs (example.com, example.org and example.net in our example) and sends a `SEARCH /cFQDN` request to each one of them, those are all search requests to the different localindxs, and the expected result is a markdown formatted string of text for each page with matches, where the matching tokens are highlighted and shown in their surrounding context.

The *Textual Content Match Set* or TCMS is the result of joining all those responses with the DEMS.

With the search *how to play the drums*, the TCMS obtained after querying all the cFQDNs in the DEMS first page could be something like this:

```
{
  "srch_id": 1,
  "page_num": 1,
  "page_size": 3,
  "total_pages": 234,
  "matches": [
    {
      "rank": 1,
      "domain": "example.com",
      "pages": [
        {
          "path": "/easy-peasy.html",
          "context": "and if you've ever wanted to learn to
                      *play the drums*, now is the best time
                      to do it. With an unlimited number of
                      online resources, it's become easier
                      than ever to pick up a pair of drum
                      sticks and start learning"
        },
        {
          "path": "/hard.html",
          "context": "Not necessarily. I don't find *drums*
                      easier than guitar or piano. Plus
                      learning rudiments and playing them
                      fast is hard compared to guitar or
                      piano or almost any other instrument
                      I've learned, mainly"
        }
      ]
    },
    {
```

```
    "rank": 2,
    "domain": "example.org",
    "pages": [
      {
        "path": "/countries/japan/",
        "context": "the Japanese taiko *drums* are also
                    *played* in this way. Next, one of
                    the drumsticks is laid on the rim
                    and it is struck using the other
                    drumstick. One can hear the crack
                    of the wood being struck. Then
                    there is the"
      }
    ]
  },
  {
    "rank": 3,
    "domain": "example.net",
    "pages": [
      {
        "path": "/2028/june/15/230748/0004501.xhtml",
        "context": "to start playing *drums*? Learning
                    how to *play* can be exciting and
                    rewarding, but there are a few
                    things that you should do before
                    you dive in. The good news is
                    that it's not as hard to learn to
                    *play* *drums* as it may"
      }
      ]
    }
  ]
}
```

Hereafter it's all about presenting those results in a friendly way to client, in the case of human interfaces, or to process them properly in bot clients.

Tenancy models

Regarding the tenancy preferences, two main types of ciprnode implementations are expected to evolve:

ST ciprnode: *single-tenant oriented* implementations, where everything is though to have the lowest possible hardware requirements and the lowest possible resource consumption. Ideal ST implementations runs smoothly in the simplest homelab, in a very light container, in a SBC or even in a Tamagotchi.

Even using embedded DBMS'[10], the main complication for this type of implementation is the size of an eventually well populated Cipr.

No matter what, STs are the ideal implementations because they guarantee distribution, independence and decentralization.

MT ciprnode: *multi-tenant oriented* implementations, where the main goal is to host multiple ciprnodes instances in a single server. This

[10]SQLite, InfinityDB, Perst...

type of implementation probably share a large DBMS/RDBMS under the hood, it's suited to handle a heavy network traffic load so, they are mostly to be deployed in large datacenters.

With this type of implementation increases the risk of centralization and the risks to the security of the Ciprsys, but their existence is justified: they facilitate more publishers having a presence in the Cipr.

Of course, there is no reason to limit the possible types of ciprnode to STs and MTs, hybrid types or totally different approaches could exist and coexist.

PoC

Later.

Epilogue

The Cipr isn't intended to replace existing search engines or directories, it can't, it's just an alternative[11]. The Ciprsys will grow to be of use if—and only if—a relevant group of domain holders—content publishers—get interested in listing their resources there, if regular search engines and directories are enough for their needs, the Cipr will be just an anecdote.

[11]A better one, certainly.

Contribution

Glad to accept PRs for all you know is wrong in here, all you know is missing and all you know can be improved.

https://codeberg.org/Cipr/specification