

Open-Search - p2p web index

<http://www.open-search.net>
preliminary draft.

Copyright © 2006, 2007 Erik Borra, Robin Gareus, Koen Martens

Revision History

Revision 4 02 Feb 2007
Draft for review.

The open-search project proposes to build a distributed, peer-to-peer, search-engine. By combining the already existing technologies of peer-to-peer file storage, distributed crawling and peer-to-peer searching, we hope to solve the problems inherent to a centralised search-engine: manipulation, censorship and profiling.

1. Preamble and Copyrights

1.1. NEWS

no news is good news.

1.2. DOWNLOAD

the open-search-agent **will be** available at Sourceforge (http://sourceforge.net/project/showfiles.php?group_id=0). Read Build and Install Instructions for further information.

1.3. LICENSE

open-search is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License Version 2 as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR

PURPOSE. See the GNU General Public License for more details.

You should receive a copy of the GNU General Public License along with the program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

gnu.org (<http://www.gnu.org/licenses/>) provides further information on-line.

2. Introduction

2.1. Overview

2.2. Motivation

3. Project Structure

3.1. Review of Existing Technologies

YaCy

yacy.de (<http://yacy.de/>) - p2p web search engine with all virtues and vices of a Java implementation. YaCy was originally designed as P2P web-proxy and virtual cyberspace on top of the Internet (TCP/IP). It offers a fairly good playground for P2P applications.

open-search and yacy differ in few design but many implementation decisions. see Table 1 . The main difference is that yacy started off as a p2p-web-cache, while open-search separates the cache and builds on top of a p2p index, addressing privacy issues.

Opensearch

opensearch.a9.com (<http://opensearch.a9.com>), opensearch.org (<http://opensearch.org>)

Opensearch is a collection of simple formats for the sharing of search results.

basically content indexing software, interesting for later stages of the open-search project

Table 1. P2P-Search Software-Application Matrix

	open-search	YaCy	GPU-search
<i>What?</i>	distributed p2p-based web indexing	distributed p2p-based web indexing	centralized p2p-based web indexing
<i>Website</i>	open-search.net (http://open-search.net)	yacy.de (http://yacy.de)	gpu.sf.net (http://gpu.sourceforge.net/search_engine.p)
<i>Implementation:</i>	none yet - modular; POSIX-C	monolithic; JAVA	modular; windows / linux-wine
<i>P2P engine</i>			GPU
<i>DHT</i>	.	.	.
<i>Crawler</i>	.	.	implements crawler only.
<i>Privacy</i>		-	-
<i>Anonymity</i>	.	.	.
<i>Notes</i>	.	bottom up design. search on top of the web-proxy. >2 yrs ahead. about 100 users. drops TLAs.	distributed crawling and indexing on top of GPU.

3.2. Goals, Requirements and Distinct Features of open-search

Goals:

- open and free: open source, cross platform, portable.
- quick search query execution
- consistent and up-to date search results
- scalable to > 10¹⁰ documents in the index
- easy to install, setup and use.

Distinct Feat:

- many.
- many more.
- stay tuned until we get there.
- low latency and privacy are good guesses for now.

3.3. Framework Structure

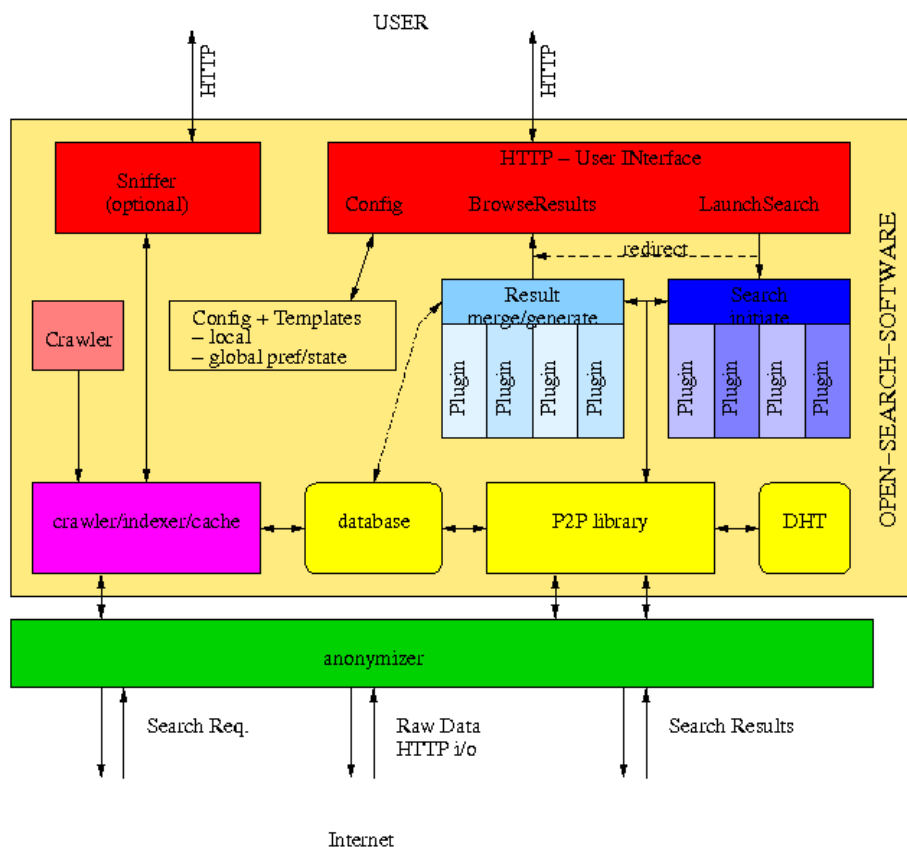


Figure 1. gives an overview of the open search agent and its major internal modules: red: user interface (http server). blue: search-core. yellow: distributed data storage.

In short: the user specifies one or more search-terms and is redirected to the result-page. Meanwhile the user's request is parsed and hashed to search-key(s), which are resolved using the open-search P2P network. Finally the returned values for each key are merged, extended (web-cache) and rendered as result entries.

3.3.1. Naming Conventions

open-search-agent

the whole software framework

server

any part within the agent that listens on a TCP/IP port

crawler

the part that does standalone crawling

proxy

optional network pass-tru, capture, caching software that harvests and indexes user-browser traffic

user

source of PEBTACs

client

the P2P network software - "network client"

front-engine

HTTP-server that manages user-search requests

backend

general term for funderlying software structure (database, P2P engine, network, content indexing,..)

web-bar

standalone UI that can be integrated in a browser

4. Functional Design (Top-Down)

4.1. Action Flow

This chapter explains the relevant steps performed by the open-search agent. This viewpoint extends Fig. 1 by listing actions that each user-search request will produce:

1. parse User-Keyword to search-keys.
2. open-search P2P get values for search-keys.
3. Merge and sort Values. build URL and Meta info lists
4. Get Summary, meta-info and return results

The steps 2-4 are pipelined and threaded. This means that the user will be presented with results early (step4) while the other processes (step 2-4) remain running and update the information. List 1 elaborates on each step.

List 1.

Step 1 - A

Access an open-search server via HTTP using a web browser

Step 1 - B (optional)

install and configure local open-search agent/server

Step 1 - B (optional)

Customize search preferences

Step 1 - C

issue a search request to the agent.

Step 1 - D

AGENT: parse search request into keys.

Step 1 - D (optional / background)

AGENT: [reverse] lookup similar search keys for this key and related pages.

Step 1 - F

AGENT: build search request for P2P engine

Step 2

P2P: Look up value(s) for the search keys. Values represent URL entries and static-meta-info for the URL.

Step 3 - A

search-result-builder: get extra info for each entry, build summary, cache,...

Step 3 - B

search-result-builder: sort and filter.

Step 3 - C

search-result-builder: feed step 1-D with related pages/keys. also feed the crawler with links.

Step 4

search-result-formatter: render results for the user, build cached-preview summaries, etc.

4.2. HTTP Daemon

A built-in webserver provides the main interface to open-search. This approach is both simple and flexible. It provides the basis to separate front-end design and back-end functionality. The HTTP protocol with with XML/HTML API is a common enough to allow various add-ons (proxy, auth, filter, styling) on top of open-search. A standard CGI interface would benefit quick prototyping, but a customized open-search-httpd will both perform better and simplify the interface to the search-plugins ¹. The user-interface itself has no functional purpose for open-search other then to provide the user and development(!) interface.

4.3. Search Plugins

To address different needs and eventually outsource development, search algorithms are modularized as plugins. However the plugin-loader and interface-core is a major part of open-search. (there will also be libraries for commonly used functions for OS-plugins). There are different types of plugins and plugins can also interact with each other (plugin-suite). From the top-view: a plugin handles a user request (and can internally call other plugins to complete the job).

Search Plugin: act on user request and schedule lookups on the P2P network.

Result render Plugin

act on user request (optional AJAX?! reload/poll/update result-page see Section 6.3)

Result merge Plugin

pre-process results from P2P network. (enqueue to render plugin)

Result search Plugin

act on special P2P network request.

4.4. Network

The p2p network is opaque! For upper layers it's a "black box" that maintains a hash-table all by itself! The Plugins just look-up data in there. later versions can also update the data (ratings, comments,..) in the P2P network, which is basically the job of the crawler.

In the first version we are planning to use some off-the shelf solution that serves as p2p network (such as gnutella, ocean-store or freenet). Later versions will incorporate a customized P2P network layer that addresses low-latency, privacy/anon. and other issues: multicasting, optimize packet size, incoming connections, ring maintenance, recursive and transitive lookups,...

There can be multiple P2P networks for different purposes. or a hybrid solution: fi. while the search-network is completely decentralized, there can be dedicated (centralized) caches, proxies or servers to efficiently parse pdfs, images, etc.

4.5. Database and Index

The database is made of several distributed hash tables: A simple 2D map to link hashes and a large p2p-tank for meta information. Here's a first Draft for the DHT Key/Value dataformat. per URL entry:

- Title
- URL

- Search-Keys (DHT KEYS)
- Search-Keys (TEXT FORMAT)
- data/time modified/crawled/indexed
- ratings (category tags and weight for each search-key)
- mirror URLs (optional)
- crawled by [host/person], indexed by [host/person]. optional pgp-signature (optional - trusted search)
- md5-sum of page (optional)
- user-comments (optional)
- Short-Excerpt/Summary (maybe not part of DHT info)

4.6. Data-format and Search-algorithms

The first and devel version will use a fixed data set and plugins to import data from external sources. open-search intends to implement a binary-compatible protocol. (config, cache, or home folders can be used on any OS). UTF-8 is the choice of character encoding.

keep in mind: URLs are unique, while search keys are not! some urls have aliases or mirrors, some are dynamic, others static. - open-search needs to transform a database of unique URLs with multiple search-keys into a Database with unique hashed-keys. Idea: find disjunct set of search-tags that allows to minimize the key-space in a karnough diagram for all URLs. URLs can be sorted into categories (used both for indexing/tagging and for crawler-priority-queueing). (example of categories can include php-script, secure-connection, mirrored-site,..)

serializing hash entries to txt files or XML-dumps is nice for import/export/transcode and debugging.

we need to investigate how much better a SQL database will perform once the dataset grows. postgres is known for very good query optimization, internal hashing and data spatialization!

4.7. Crawler, Proxy

Each open-search agent has two internal queues/buffers: one for URL to be downloaded for indexing. and two, a buffer with the retrieved data.

An (optional/no privacy) proxy processes can be used to feed user-browsed pages directly into the process-buffer. It can also parse links and learn/rate URLs from user paths (referrer). or be feed crawled pages into the private/personal cache, ie. the opensearch client as personal cache thing, prefetching pages during browsing

The crawler is trivial, although some efforts could be taken to address privacy on crawler level. Crawling could happen time-delayed, or semi-randomly, schedule random crawls on neighbor hosts(!? security), or even drop pages on heuristic limits. Also important is to pay attention to IP proto priority for the crawler (don't suck up user bandwidth, but beware of IP filters!). the open-search proxy seems to be the right place for pluggable session-level privacy.

4.8. Indexing and parsing

The open-search indexer-core checks document-type and URL/Hash from the index-buffer-stack and sorts them into *channels*, which are processed by [different] index-plugins. The return value of the plugins are standardized hashes to be fed to the P2P network-storage.

The indexer can also schedule (enqueue) new URLs for indexing. a priority system can be used to flush or process the channels/queues (ie. look-up the last-crawled date first).

The resulting database is a collection of URLs + meta information, linked to search-keys. Those keys need to be parsed from the page and weighted for the given URL.

4.9. Privacy and Anonymity

As the agent is open-source, we can not guarantee anonymity if it is build on top of the network. User anonymization needs to be implemented below or inside the P2P network. Anonymizing introduces various disadvantages: eg. extra latency. open-search aims to be compatible to tor as optional anonymizer. Privacy issues are addressed at application/session layer with the goal to obfuscate user habits and eliminate individual statistics, but they can not assure anonymity.

5. Design and Implementation (bottom-up)

As of writing this document, the implementation is in pre-alpha state. Nevertheless present tense is used throughout this documentation.

5.1. Open-Search Framework

This chapter will describe the abstract building blocks bottom up.

5.2. External Interface Definitions

5.3. Internal Interfaces

5.4. Communication Protocols

5.5. Configuration

open-search requires a lot of configuration that is rarely used and a few config-values that are used a lot!. Configuration is subdivided in a "running-config" and a "boot-config" both of which are roots of the configuration tree structure.

The config-tree is serializable (txt,xml,YAML) and can be edited via a HTTP front-end. Internally sub-trees can stored in binary format and cached for frequent use (MESI cache).

configuration should be standard / expert (conf. category, level), so that the average user won't be bothered with settings they don't understand or care about.

6. The Agent - Developer Information

Here's how we get started:

- Hijack a HTTP server (microhttpd, dhttpd,..)
 - template (XSLT) + CGI
 - CGI - open-search config interface
 - CGI - open-search search + result interface
- Plugin Loader
 - Launch Search
 - Merge (dummy) results
 - Hash/Key Data-format (wrapper)
- P2P Plugins

6.1. The Open Search Agent

Brainstorm:

- Start with HTTPd and configuration interface! branch off microhttpd, minihttpd, dhttpd, boa or even lighttpd...
- proxy-modules might become too complex -> off-the-shelf solutions? maybe <http://tinyproxy.sourceforge.net>
- the crawler / indexer / proxy could be a perl script (in the first version) it allows to try out ideas and experiment quickly - perl is preinstalled on Linux and osX but we need some uncommon perl-modules so we should mirror/ship them.. there is perl for windows,too. and even more important: there are ready-made perl modules for crawling, HTML-parsing, indexing and summarizing HTML/pdf, extracting links, etc.
- Crawler / indexer / proxy: libcurl, libhtml, libcgi,. offer similar functionality in C - better support in the long run, less bugs, easier to maintain (!?), more complicated to make (small) changes.
- crawler- and indexer-obscurity is easier to do than hiding search/browsing stats. and structure of P2P plus low latency favors reliable communication (transitive lookups) and thus do not allow to introduce query-source obfuscation on application level. The best idea so far is to implement fake query modification and filtering in the local search-key-hash parser.
- p2p data storage: 1st use ocean-store (<http://oceanstore.cs.berkeley.edu/>), freenet (<http://freenetproject.org/>) or gnutella (<http://en.wikipedia.org/wiki/Gnutella>). write plugins to import/export internal data structure(s). Later in the project we will switch to access the underlying DHT directly: bamboo-dht, libgnutella. Chord, pastry/tapestry, Leopard, k-Ary-DHT,..
- The first versions (bootstrap) will "fake" the P2P network to get a useable front-end for testing. Traffic estimations from the "dummy P2P network" can be used to simulate P2P scenarios! (first idea to collect stats: share database via NFS. write IP + file-inode logs.)
- .

6.2. Libraries and Classes

6.3. Front-end interface and templates.

search results can be updated dynamically by using AJAX techniques, or static reloads! There will be both non-blocking and blocking I/O! The HTTP-client (javascript) specifies a request mode:

```
[poll,read]? [nonblock|blocktimeout=NN]?
```

where NN is 1/10 seconds.

7. The Agent - User's Guide

7.1. Build and Install Instructions

7.1.1. System Requirements

open-search is currently known to work on gnu/Linux and Mac OS-X.

7.1.2. Binaries and Sources

open-search-agent is being developed.. there is no public release yet.

Notes

1. this interface could be implemented on top of cgi; but cgi-sessions ususally terminate after each request.
2. there's come configuration API for the DHT, too.