Data acquisition, extraction, and storage

Web content acquisition

Pierre Senellart

21 September 2023

Outline

Internet and the Web

Internet: physical network of computers (or hosts)

World Wide Web, Web, WWW: logical collection of hyperlinked documents

- static and dynamic
- public Web and private Webs
- each document (or Web page, or resource) identified by a URL
- the largest collection of information every built, most readily accessible!

The Internet protocol stack

A stack of communication protocols, on top of each other.

Application	HTTP, FTP, SMTP, DNS	
Security	SSL/TLS	(encryption, authentication)
Transport	TCP, UDP, ICMP	(sessions, reliability...)
Network	IP (v4, v6)	(routing, addressing)
Link	Ethernet, 802.11 (ARP)	(addressing local machines)
Physical	Ethernet, 802.11 (physical)	

IP (Internet Protocol) [IETF, 1981a]

- Addressing machines and routing over the Internet
- Two versions of the IP protocol on the Internet: IPv4 (very well spread) and IPv6 (not that well-spread yet)
- IPv4: 4-byte addresses assigned to each computer, e.g., 137.194.2.24. Institutions are given ranges of such addresses, to assign as they will.
- Problem: only 2^{32} possible addresses (actually, a large number of them cannot be assigned to new hosts, for multiple reasons). This means many hosts connected to the Internet do not have an IPv4 address and some network address translation (NAT) occurs.
- IPv6: 16-byte addresses; much larger address space! Addresses look like 2001:660:330f:2::18 (meaning 2001:0660:0330f:0002:0000:0000:0000:0018). Other nice features (multicast, autoconfiguration, etc.).

TCP (Transmission Control Protocol) [IETF, 1981b]

- One of the two main transport protocols used on IP, with UDP (User Datagram Protocol)
- Contrarily to UDP, provides reliable transmission of data (acknowledgments)
- Data is divided into small datagrams that are sent over the network, and possibly reordered at the end point
- Like UDP, each TCP transmission indicates a source and a destination port number (between 0 and 65535) to distinguish it from other traffic
- A client usually select a random port number for establishing a connection to a fixed port number on a server
- The port number on a server conventionally identifies an application protocol on top of TCP/IP: 22 for SSH, 25 for SMTP, 110 for POP3...

DNS (Domain Name System) [IETF, 1999a]

- IPv4 addresses are hard to memorize, and a given service (e.g., a Web site) may change IP addresses (e.g., new Internet service provider)
- Even more so for IPv6 addresses!
- DNS: a UDP/IP-based protocol for associating human-friendly names (e.g., `www.google.com`, `weather.yahoo.com`) to IP addresses
- Hierarchical domain names: `com` is a top-level domain (TLD), `yahoo.com` is a subdomain thereof, etc.
- Hierarchical domain name resolution: root servers with fixed IPs know who is in charge of TLDs, servers in charge of a domain know who is in charge of a subdomain, etc.
- Nothing magic with `www.google.com`: just a subdomain of `google.com`.

URL (Uniform Resource Locator) [IETF, 1994]

$\underbrace{\texttt{https}}_{\text{scheme}}\texttt{://}\underbrace{\texttt{www.example.com}}_{\text{hostname}}\underbrace{\texttt{:443}}_{\text{port}}\texttt{/}\underbrace{\texttt{path/to/doc}}_{\text{path}}\underbrace{\texttt{?name=foo\&town=bar}}_{\text{query string}}\underbrace{\texttt{\#para}}_{\text{fragment}}$

scheme: way the resource can be accessed; generally http or https

hostname: domain name of a host (cf. DNS); hostname of a website may start with `www.`, but not a rule.

port: TCP port; defaults: 80 for `http` and 443 for `https`

path: logical path of the document

query string: optional additional parameters (dynamic documents)

fragment: optional subpart of the document

Relative URLs with respect to a context (e.g., the URL above):

`/titi`	`https://www.example.com/titi`
`tata`	`https://www.example.com/path/to/tata`

The Web: a mixture of technologies

- For content: HTML, but also PDF, Word documents, text files, JSON documents, XML (RSS, SVG, MathML, etc.)...
- For presenting this content: CSS, XSLT
- For animating this content: JavaScript, AJAX, JavaScript libraries (jQuery, etc.)...
- For interaction-rich content: `<canvas>`, WebGL...
- And on the server side: a Web server program (Apache, nginx, Tomcat, IIS, Node.js, etc.), any programming language and database technology to serve this content, e.g., PHP, Java servlets, Scala, Python, JavaScript, etc. with associated Web programming frameworks

Quite complex to manage! Being a Web developer nowadays requires mastering a lot of different technologies. Exploiting Web content may require being able to handle a lot of different technologies!

Outline

HTML (HyperText Markup Language) [W3C, 2014]

- normalized by the W3C (World Wide Web Consortium) formed of industrials (Microsoft, Google, Apple...) and academic institutions (ERCIM, MIT, etc.) in concert with another group of industrials, the WHATWG group (Apple, Mozilla, Google, Microsoft) representing Web browsers
- open format: possible processing by a wide variety of software and hardware
- text files with tags
- describes the structure and content of a document, focus on accessibility
- (theoretically) no presentation information (this is the role of CSS)
- no description of dynamic behaviors (this is the role of server-side languages, JavaScript, etc.)

The HTML language

- HTML is a language alternating text and tags (`<blabla>` or `</blabla>`)
  - Tags allow structuring each part of a document, and are used for instance by a browser to lay out the document.
- HTML files
  - are structured in two main parts: the header `<head> ... </head>`) and the body `<body> ... </body>`)
- In HTML, blanks (spaces, tabs, carriage returns) are generally equivalent and only serve to delimit words, tags, etc. The number of blanks does not matter.

Tags

- Syntax: (opening and closing tag)

```
<tag attributes>content</tag>
```

or (element with no content)

```
<tag attributes>
```

tag keyword referring to some particular HTML element

content may contain text and other tags

attributes represent the various parameters associated with the element, as a list of `name="value"` or `name='value'`, separated by spaces (quotes are not always mandatory, but they become mandatory if `value` has “exotic” characters)

Tags

- Names of elements and attributes are usually written in lowercase, but `<head>` and `<HeAd>` are equivalent.
- Tags are opened and closed in the right order (`<b><i></i></b>` and not `<b><i></b></i>`).
- Strict rules specify which tags can be used inside which.
- Under some conditions, a tag can be implicitly closed, but these conditions are complex to describe.
- `<!--foobar-->` denotes a comment, which is not to be interpreted by a Web client.

Structure of a document

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Header of the document -->
  </head>
  <body>
    <!-- Body of the document -->
  </body>
</html>
```

- The doctype declaration `<!DOCTYPE ...>` specify which HTML version is used, here HTML5.
- The language of the document is specified with the `lang` attribute of the main `<html>` tag.

Header

- The header of a document is delimited by the tags `<head>... </head>`.
- The header contains meta-informations about the document, such as its title, encoding, associated CSS and JavaScript files, etc. The two most important items are:
  - The character set of the page, usually at the very beginning of the header

    ```
    <meta charset="utf-8">
    ```

  - The title of the page (the only required item inside the header). This is the information displayed in the title bar of Web browsers.

    ```
    <title>My great website</title>
    ```

Character sets

Unicode: character repertoire, assigning to each character, whatever its script or language, an integer number.

Examples

A	$\rightarrow$	65	ε	$\rightarrow$	949
é	$\rightarrow$	233	$\aleph$	$\rightarrow$	1488

Character set: concrete method for representing a Unicode character.

Examples (é)

iso-8859-1	11101001		only for some characters
utf-8	11000011	10101001	
utf-16	11101001	00000000	

utf-8 has the advantage of being able to represent all Unicode characters, in a way compatible with the legacy ASCII encoding.

The body of a HTML document

- `<body>... </body>` tags delimit the body of a document.
- The body is structured into sections, paragraphs, lists, etc.
- 6 tags describe sections, by decreasing order of importance:
  - `<h1>Title of the page</h1>`
  - `<h2>Title of a main section</h2>`
  - `<h3>Title of a subsection</h3>`
  - `<h3>Title of a subsubsection</h3>`
  - ...
- `<p> ... </p>` tags delimit paragraphs of text. All text paragraphs should be delimited thusly.
- Directly inside `<body>... </body>` can only appear block elements: `<p>`, `<h1>`, `<form>`, `<hr>`, `<ul>`, `<table>`... in addition to the `<div>` tag which denotes a block without precise semantics.

Links

- What differetiates Web pages (hypertext pages) from normal documents: links!
- Introduced with `<a> ... </a>`
- Navigating a link can bring to:
  - a resource on another server or another file of the same server
  - another part of the same document

Links

Links are made using the `href` attribute of the `<a>` tag, whose content will be the link:

```
<a href="http://www.cnrs.fr/">
  <img src="images/cnrs.gif" alt="CNRS">
</a>

<a href="bio/indexbioinfo.html">Bioinformatics</a>
```

Anchors

- Anchors serve to reach a precise point in the document.
  - They are defined, either on an existing tag by using the `id` attribute, or with an `<a id="...">` :

```
<h3 id="tutorials">Tutorials</h3>
<a id="tutorials">
```

  - Then, one can link to this anchor:

```
<a href="#tutorials">tutorials</a>
<a href="http://www.w3.org/#tutorials">tutorials</a>
```

  - Commonly, the old `<a name="...">` syntax is used.

The different versions of HTML

- HTML5 (2014 and onwards): "living" standard, continuously updated
- HTML 4.01 (1999): historically the main standard before HTML5 arose
- XHTML (2000): an attempt of XML-ifying HTML, making it stricter, mostly a failure (though HTML5 also now has valid XML syntax variant)
- XHTML 1.1 and XHTML 2.0: complete failures, unusable and unused

Tag soup

- Quite a few HTML documents on the Web date back from before HTML 4.01!
- In practice: many Web pages do not respect any standards at all (with or without doctype declarations) $\Longrightarrow$ browsers do not respect these standards $\Longrightarrow$ tag soup!
- When dealing with pages from the real Web, necessary to use all sorts of heuristics to interpret a Web page.

HTML vs XHTML

- XHTML: an XML format
- Tags without content `<img>`, are written `<img />` in XHTML.
- Some elements can be left unclosed in HTML (`<ol> <li> one <li>two </ol>`), but closing is mandatory in XHTML.
- Attribute values can be written without quotes (`<img src=toto.png alt=toto>`) in HTML, quotes are required in XHTML.
- Element and attribute names are not case-sensitive in HTML (`<HTMl laNg=fr>`), but are in XHTML (everything must be in lowercase).
- Attributes `xmlns` and `xml:lang` on the `<html>` tag in XHTML.
- And some other small subtleties. . .

Outline

HTTP (HyperText Transfer Protocol) [IETF, 1999b]

- Application protocol at the basis of the World Wide Web
- Most widely used version: HTTP/1.1 (though HTTP/2 exists and HTTP/3 is a proposed standard)

- Client request:

```
GET /MarkUp/ HTTP/1.1
Host: www.w3.org
```

- Server response:

```
HTTP/1.1 200 OK
...
Content-Type: text/html; charset=utf-8

<!DOCTYPE html ...> ...
```

- Two main HTTP methods: GET and POST (HEAD is also used in place of GET, to retrieve meta-information only).
- Additional headers, in the request and the response
- Possible to send parameters in the request (key/value pairs).

GET

- Simplest type of request.
- Possible parameter are sent at the end of a URL, after a '?'
- Not applicable when there are too many parameters, or when their values are too long.
- Method used when a URL is directly accessed in a browser, when a link is followed, and for some forms.

Example (Google query)

URL: `http://www.google.com/search?q=hello`

Corresponding HTTP GET request:

```
GET /search?q=hello HTTP/1.1
Host: www.google.com
```

POST

- Method only used for submitting forms.

Example

```
POST /php/test.php HTTP/1.1
Host: www.w3.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 100

type=search&title=The+Dictator&format=long&country=US
```

Parameter encoding

- By default, parameters are sent (with GET or POST) in the form: `name1=value1&name2=value2`, and special characters (accented characters, spaces...) are replaced by codes such as +, %20. This way of sending parameters is called `application/x-www-form-urlencoded`.
- For the POST method, another heavier encoding can be used (several lines per parameter), similar to the way emails are built: mostly useful for sending large quantity of information. Encoding named `multipart/form-data`.

Status codes

- The HTTP response always starts with a status code with three digits, followed by a human-readable message (e.g., 200 OK).
- The first digit indicates the class of the response:
  1. Information
  2. Success
  3. Redirection
  4. Client-side error
  5. Server-side error

Most common status codes

200	OK
301	Permanent redirection
302	Temporary redirection
304	No modification
400	Invalid request
401	Unauthorized
403	Forbidden
404	Not found
500	Server error

Virtual hosts

- Different domain names can refer to the same IP address, i.e., the same physical machine (e.g., `www.google.fr` and `www.google.com`)
- When a machine is contacted by TCP/IP, it is through its IP address
- No *a priori* way to know which precise domain name to contact
- In order to serve different content according to the domain name (virtual host): header `Host:` in the request (only header really required)

Example

```
GET /search?hl=fr&q=hello HTTP/1.1
Host: www.google.fr
```

Content type

- The browser behaves differently depending on the content type returned: display a Web page with the layout engine, display an image, load an external application, etc.
- MIME classification of content types (e.g., `image/jpeg`, `text/plain`, `text/html`, `application/xhtml+xml`, `application/pdf` etc.)
- For a HTML page, or for text, the browser must also know what character set is used (this has precedence over the information contained in the document itself)
- Also returned: the content length (can be used to display a progress bar)

Example

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 3046
```

Client and server identification

- Web clients and servers can identify themselves with a character string
- Useful to serve different content to different browsers, detect robots. . .
- . . . but any client can say it's any other client!
- Historical confusion on naming: all common browsers identify themselves as Mozilla!

Example

```
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; fr;
rv:1.9.0.3) Gecko/2008092510 Ubuntu/8.04 (hardy)
Firefox/3.0.3

Server: Apache/2.0.59 (Unix) mod_ssl/2.0.59 OpenSSL/0.9.8e
PHP/5.2.3
```

Content negotiation

- A Web client can specify to the Web server:
  - the content type it can process (text, images, multimedia content), with preferrence indicators
  - the languages preferred by the user
- The Web server can thus propose different file formats, in different languages.
- In practice, content negociation on the language works, and is used, but content negociation on file types does not work because of bad default configuration of some browsers.

Example

```
Accept: text/html,application/xhtml+xml,application/xml;
q=0.9,*/*;q=0.8
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
```

Cookies [IETF, 2000]

- Information, as key/value pairs, that a Web server asks a Web client to keep and retransmit with each HTTP request (for a given domain name).
- Can be used to keep information on a user as she is visiting a Web site, between visits, etc.: electronic cart, identifier, and so on.
- Practically speaking, most often only stores a session identifier, connected, on the server side, to all session information (connected or not, user name, data...)
- Simulates the notion of session, absent from HTTP itself

Example

```
Set-Cookie: session-token=RJYBsG//azkfZrRazQ3SPQhlo1FpkQka2;
path=/; domain=.amazon.de;
expires=Fri Oct 17 09:35:04 2008 GMT

Cookie: session-token=RJYBsG//azkfZrRazQ3SPQhlo1FpkQka2
```

Originating URL

- When a Web browser follows a link or submits a form, it transmits the originating URL to the destination Web server.
- Even if it is not on the same server!

Example

```
Referer: http://www.google.fr/
```

Outline

Web Crawlers

- crawlers, (Web) spiders, (Web) robots: autonomous user agents that retrieve pages from the Web
- Basics of crawling:
  1. Start from a given URL or set of URLs
  2. Retrieve and process the corresponding page
  3. Discover new URLs (cf. next slide)
  4. Repeat on each found URL
- No real termination condition (virtual unlimited number of Web pages!)
- Graph-browsing problem

  deep-first: not very adapted, possibility of being lost in robot traps

  breadth-first

  combination of both: breadth-first with limited-depth deep-first on each discovered website

Sources of new URLs

- From HTML pages:
  - hyperlinks `<a href="...">...</a>`
  - media `<img src="...">` `<embed src="...">` `<object data="...">`
  - frames `<frame src="...">` `<iframe src="...">`
  - JavaScript links `window.open("...")`
  - etc.
- Other hyperlinked content (e.g., PDF files)
- Non-hyperlinked URLs that appear anywhere on the Web (in HTML text, text files, etc.): use regular expressions to extract them
- Referrer URLs
- Sitemaps [sitemaps.org, 2008]

Scope of a crawler

- Web-scale
  - The Web is infinite! Avoid robot traps by putting depth or page number limits on each Web server
  - Focus on important pages [Abiteboul et al., 2003]
- Web servers under a list of DNS domains: easy filtering of URLs
- A given topic: focused crawling techniques [Chakrabarti et al., 1999, Diligenti et al., 2000, Gouriten et al., 2014] based on classifiers of Web page content and predictors of the interest of a link.
- The national Web (cf. public deposit, national libraries): what is this? [Abiteboul et al., 2002]
- A given Web site: what is a Web site? [Senellart, 2005]

Outline

A word about hashing

Definition

A hash function is a deterministic mathematical function transforming objects (numbers, character strings, binary…) into fixed-size, seemingly random, numbers. The more random the transformation is, the better.

Example

Java hash function for the `String` class:

$$\sum_{i=0}^{n-1} s_i \times 31^{n-i-1} \bmod 2^{32}$$

where s_i is the (Unicode) code of character i of a string s.

Identification of duplicate Web pages

Problem

Identifying duplicates or near-duplicates on the Web to prevent multiple indexing

trivial duplicates: same resource at the same canonized URL:

```
http://example.com:80/toto
http://example.com/titi/../toto
```

exact duplicates: identification by hashing

near-duplicates: (timestamps, tip of the day, etc.) more complex!

Near-duplicate detection

Edit distance. Count the minimum number of basic modifications (additions or deletions of characters or words, etc.) to obtain a document from another one. Good measure of similarity, and can be computed in $O(mn)$ where m and n are the size of the documents. But: does not scale to a large collection of documents (unreasonable to compute for every pair!).

Shingles. Idea: two documents similar if they mostly share the same succession of k-grams (succession of tokens of length k).

Example

I like to watch the sun set with my friend.
My friend and I like to watch the sun set.

$S =${i like, like to, my friend, set with, sun set, the sun, to watch, watch the, with my}
$T =${and i, friend and, i like, like to, my friend, sun set, the sun, to watch, watch the}

Hashing shingles to detect duplicates [Broder et al., 1997]

- Similarity: Jaccard coefficient on the set of shingles:

$$J(S, T) = \frac{|S \cap T|}{|S \cup T|}$$

- Still costly to compute! But can be approximated as follows:
  1. Choose N different hash functions
  2. For each hash function h_i and each set of shingles $S_k = \{s_{k1} \dots s_{kn}\}$, store $\phi_{ik} = \min_j h_i(s_{kj})$
  3. Approximate $J(S_k, S_l)$ as the proportion of ϕ_{ik} and ϕ_{il} that are equal
- Possibly to repeat in a hierarchical way with super-shingles (we are only interested in very similar documents)

Outline

Crawling ethics

- Standard for robot exclusion: robots.txt at the root of a Web server [Koster, 1994].

```
User-agent: *
Allow: /searchhistory/
Disallow: /search
```

- Per-page exclusion.

```
<meta name="ROBOTS" content="NOINDEX,NOFOLLOW">
```

- Per-link exclusion.

```
<a href="toto.html" rel="nofollow">Toto</a>
```

- Avoid Denial Of Service (DOS), wait ≈1s between two repeated requests to the same Web server

Legal aspects (France) – 1/2

- General principles:
  - to access or keep access to a "system for automated data processing" *in a fraudulous manner* is punished of two years of prison and 60 000 euros fine (Code pénal 323-1, modified by law 2015-912 on "Renseignement")
  - to disrupt the functioning of a "system for automated data processing" is punished of five years of prison and 150 000 euros fine, extended to seven years and 300 000 euros when the system is a public one containing personal information (Code pénal 323-2, modified by law 2015-912 on "Renseignement")
- A Web site hosted in a different country may invoke completely different legal principles, under a different jurisdiction
- Crawling content can be considered accessing and keeping access to a "system for automated data processing" (Cour d'appel de Paris, 5 February 2014, "Bluetouff case")

Legal aspects (France) – 2/2

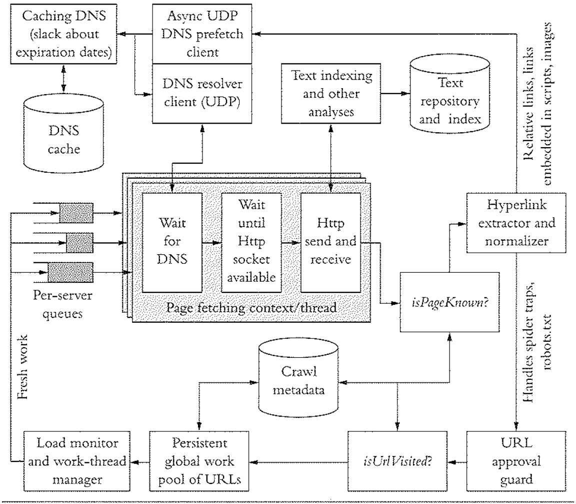
- robots.txt files are a de facto standard, and instructions in robots.txt files a receivable way to specify what can be crawled (Cour d'appel de Paris, 26 January 2011, Google vs SAIF)
- Frequent requests to a Web site can be considered as a way to disrupt the functioning of a "system for automated data processing" (Cour d'appel de Bordeaux, 15 November 2011, Cédric M. vs C-Discount), but only if it reaches abusive levels and can be shown to have cause disruption
- Web content is subject to "droit d'auteur" (Code de la propriété intellectuelle, Première partie, Livre Ier) and cannot generally be broadcast by third-parties; only transient copies are allowed (CJEU, 5 June 2014, PRCA vs NLA)
- Web content containing personal data is even more sensitive (GDPR): personal data should be collected for a specific purpose, kept updated, and protected

Parallel processing

Network delays, waits between requests:

- Per-server queue of URLs
- Parallel processing of requests to different hosts:
  - multi-threaded programming
  - asynchronous inputs and outputs (`select`, classes from `java.util.concurrent`): less overhead
- Use of keep-alive to reduce connexion overheads

General Architecture [Chakrabarti, 2003]



Refreshing URLs

- Content on the Web changes
- Different change rates:

  online newspaper main page: every hour or so
  published article: virtually no change
- Continuous crawling, and identification of change rates for adaptive crawling

Importance of Timely Crawling

- The Web is very volatile, with a typical half-life of URLs of a few years [Koehler, 2003]
- For many purposes (archiving, analytics), a crawl quality can be measured by its temporal coherence [Spaniol et al., 2009]
- Ideally, Web pages pointed to by a Web page should be crawled at the same time. Unrealistic in practice.
- Crawling takes time and consumes resources:
  - Limited bandwidth, limiting computing power on the crawling side
  - Because of crawling ethics, crawling a 5 million page site takes around 2 months!
  - Limitations of social networking APIs drastic: on Twitter, using the *Recent search* endpoint, at most 3 000 tweets per minute; other endpoints exist, but they also have severe limitations;

Importance of Timely Crawling

- The Web is very volatile, with a typical half-life of URLs of a few years [Koehler, 2003]
- For many purposes (archiving, analytics), a crawl quality can be measured by its temporal coherence [Spaniol et al., 2009]
- Ideally, Web pages pointed to by a Web page should be crawled at the same time. Unrealistic in practice.
- Crawling takes time and consumes resources:
  - Limited bandwidth, limiting computing power on the crawling side
  - Because of crawling ethics, crawling a 5 million page site takes around 2 months!
  - Limitations of social networking APIs drastic: on Twitter, using the *Recent search* endpoint, at most 3 000 tweets per minute; other endpoints exist, but they also have severe limitations; more than 350 000 new tweets per minute on average!

Outline

Crawling Modern Web Sites

- Some modern Web sites only work when cookies are activated (session cookies), or when JavaScript code is interpreted
- Regular Web crawlers (wget, Heritrix, Apache Nutch) do not usually perform any cookie management and do not interpret JavaScript code
- Crawling of some Websites therefore require more advanced tools

Advanced crawling tools

Web scraping frameworks such as scrapy (Python) or WWW::Mechanize (Perl) simulate a Web browser interaction and cookie management (but no JS interpretation)

Headless browsers such as htmlunit simulate a Web browser, including simple JavaScript processing

Browser instrumentors such as Selenium allow full instrumentation of a regular Web browser (Chrome, Firefox)

Proxys such as mitmproxy capable of recording and replaying a complex set of HTTP requests

OXPath: a full-fledged navigation and extraction language for complex Web sites [Sellers et al., 2011]; unfortunately deprecated

Outline

Templated Web Site

- Many Web sites (especially, Web forums, blogs) use one of a few content management systems (CMS)
- Web sites that use the same CMS will be similarly structured, present a similar layout, etc.
- Information is somewhat structured in CMSs: publication date, author, tags, forums, threads, etc.
- Some structure differences may exist when Web sites use different versions, or different themes, of a CMS

Crawling CMS-Based Web Sites

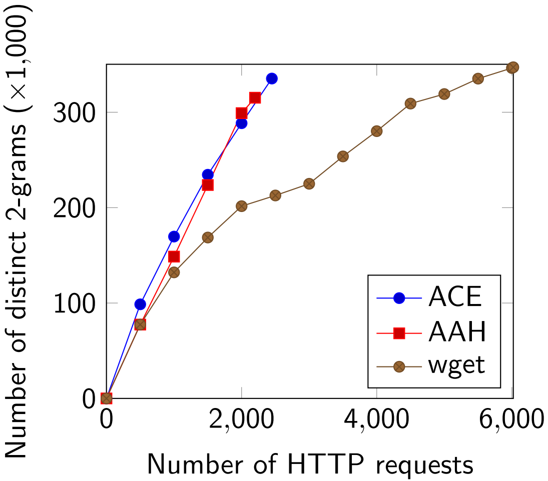
- Traditional crawling approaches crawl Web sites independently of the nature of the sites and of their CMS
- When the CMS is known:
  - Potential for much more efficient crawling strategies (avoid pages with redundant information, uninformative pages, etc.)
  - Potential for automatic extraction of structured content
- Two ways of approaching the problem:
  - Have a handcrafted knowledge base of known CMSs, their characteristics, how to crawl and extract information [Faheem and Senellart, 2013b,a] (AAH)
  - Automatically infer the best way to crawl a given CMS [Faheem and Senellart, 2014] (ACE)
- Need to be robust w.r.t. template change

Detecting CMSs

- One main challenge in intelligent crawling and content extraction is to identify the CMS and then perform the best crawling strategy accordingly
- Detecting CMS using:
  1. URL patterns,
  2. HTTP metadata,
  3. textual content,
  4. XPath patterns, etc.
- These can be manually described (AAH), or automatically inferred (ACE)
- For instance the vBulletin Web forum content management system, that can be identified by searching for a reference to a `vbulletin_global.js` JavaScript script by using a simple `//script/@src` XPath expression.

Crawling http://www.rockamring-blog.de/

[Faheem and Senellart, 2014]



Outline

Social data on the Web

Huge numbers of active users of social networking sites (2020):

Facebook	2.7 billion
YouTube	2.0 billion
TikTok+DouYin	1.3 billion
WeChat	1.2 billion
Instagram	1.2 billion
Weibo	520 million
QZone	520 million
Reddit	430 million
Kuaishou	430 million
Pinterest	416 million
Twitter	350 million

Social data on the Web

Huge numbers of active users of social networking sites (2020):

Facebook	2.7 billion
YouTube	2.0 billion
TikTok+DouYin	1.3 billion
WeChat	1.2 billion
Instagram	1.2 billion
Weibo	520 million
QZone	520 million
Reddit	430 million
Kuaishou	430 million
Pinterest	416 million
Twitter	350 million

Huge volume of shared data:

800 million tweets per day on Twitter (10,000 per second on average!)...

... including statements by heads of states, revelations of political activists, etc.

Crawling Social Networks

- Theoretically possible to crawl social networking sites using a regular Web crawler
- May be impossible: `https://www.facebook.com/robots.txt`
- Often very inefficient, considering politeness constraints
- Better solution: Use provided social networking APIs
  `https://developer.twitter.com/en/docs/api-reference-index`
  `https://developers.facebook.com/docs/graph-api/reference/`
  `https://docs.microsoft.com/en-us/linkedin/`
  `https://developers.google.com/youtube/`
- Also possible to buy access to the data, directly from the social network or from brokers. See, e.g., `https://developer.twitter.com/en/products/twitter-api/enterprise`

Social Networking APIs

- Most social networking Web sites (and some other kinds of Web sites) provide APIs to effectively access their content
- Usually a RESTful API, occasionally SOAP-baed
- Usually requires a token identifying the application using the API, sometimes a cryptographic signature as well
- May access the API as an authenticated user of the social network, or as an external party
- APIs seriously limit the rate of requests: `https://developer.twitter.com/en/docs/twitter-api/rate-limits`

REST

- Mode of interaction with a Web service
- Follow the KISS (Keep it Simple, Stupid) principle
- Each request to the service is a simple HTTP GET method
- Base URL is the URL of the service
- Parameters of the service are sent as HTTP parameters (in the URL)
- HTTP response code indicates success or failure
- Response contains structured output, usually as JSON or XML
- No side effect, each request independent of previous ones

Cross-Network Crawling

- Often useful to combine results from different social networks
- Numerous libraries facilitating SN API accesses (tweepy, Facebook4J...) incompatible with each other... Some efforts at generic APIs (OneAll, APIBlender [Gouriten and Senellart, 2012])
- Example use case: No API to get all check-ins from FourSquare, but a number of check-ins are available on Twitter; given results of Twitter Search/Streaming, use FourSquare API to get information about check-in locations.

Outline

The Deep Web

Definition (Deep Web, Hidden Web, Invisible Web)

All the content on the Web that is not directly accessible through hyperlinks. In particular: HTML forms, Web services.

Size estimate: 500 times more content than on the surface Web! [BrightPlanet, 2000]. Hundreds of thousands of deep Web databases [Chang et al., 2004]

Sources of the Deep Web

Example

- *Yellow Pages* and other directories;
- Library catalogs;
- Weather services;
- US Census Bureau data;
- etc.

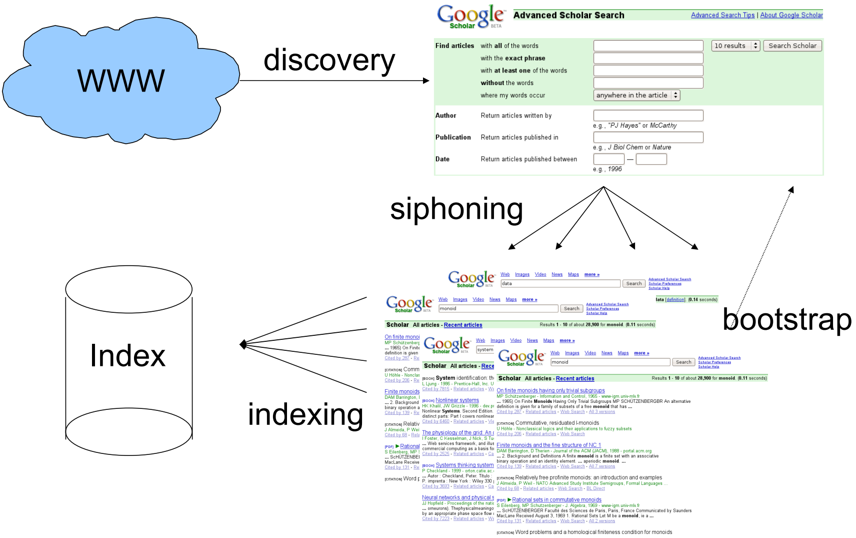
Discovering knowledge from the deep Web

[Nayak et al., 2012]

- Content of the deep Web hidden to classical Web search engines (they just follow links)
- But very valuable and high quality!
- Even services allowing access through the surface Web (e.g., e-commerce) have more semantics when accessed from the deep Web
- How to benefit from this information?
- How to analyze, extract and model this information?

Focus here: Automatic, unsupervised, methods, for a given domain of interest

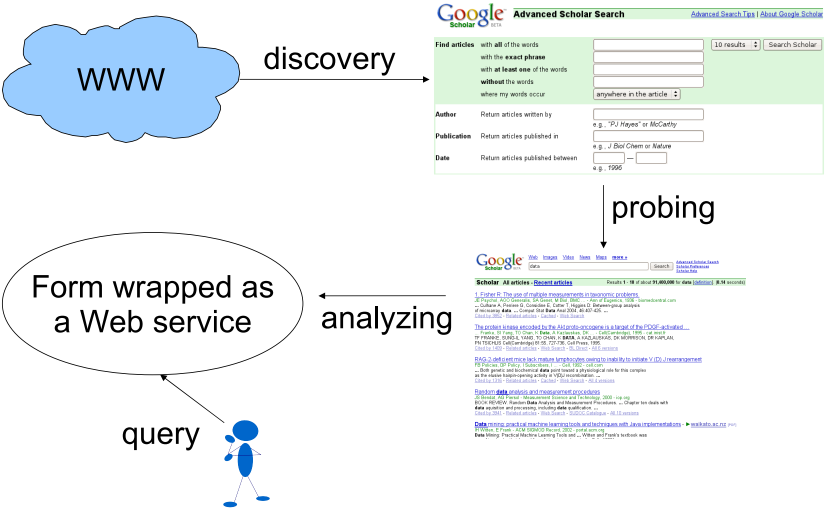
Extensional Approach



Notes on the Extensional Approach

- Main issues:
  - Discovering services
  - Choosing appropriate data to submit forms
  - Use of data found in result pages to bootstrap the siphoning process
  - Ensure good coverage of the database
- Approach favored by Google, used in production [Madhavan et al., 2006]
- Not always feasible (huge load on Web servers)

Intensional Approach



Notes on the Intensional Approach

- More ambitious [Chang et al., 2005, Senellart et al., 2008]
- Main issues:
  - Discovering services
  - Understanding the structure and semantics of a form
  - Understanding the structure and semantics of result pages
  - Semantic analysis of the service as a whole
  - Query rewriting using the services
- No significant load imposed on Web servers

Outline

Conclusion

What you should remember

- Crawling as a graph-browsing problem.
- Shingling for identifying duplicates.
- Numerous engineering issues in building a Web-scale crawler.
- Crawling modern Web content is not as easy as launching a traditional Web crawler
- Often critical to focus the crawl towards content of interest
- Ideally: a traditional large-scale crawler that knows when to delegate to more specialized crawling mechanisms (tools querying social networking APIs, deep Web crawlers, JS-aware crawlers, etc.)
- Huge variety of tools, techniques, suitable for different needs

References

Free software

wget simple yet effective Web spider

Heritrix Web-scale highly configurable Web crawler, used by the Internet Archive

Beautiful Soup Python module for parsing real-world Web pages

Scrapy rich Python module for Web crawling and content extraction

Selenium browser instrumentor, with API in several languages

To go further

- A good textbook [Chakrabarti, 2003]
- Main references:
  - HTML5 living standard [W3C, 2014]
  - HTTP/1.1 RFC [IETF, 1999b]

Bibliography I

Serge Abiteboul, Grégory Cobena, Julien Masanès, and Gerald Sedrati. A first experience in archiving the French Web. In *Proc. ECDL*, Roma, Italie, September 2002.

Serge Abiteboul, Mihai Preda, and Gregory Cobena. Adaptive on-line page importance computation. In *Proc. WWW*, May 2003.

BrightPlanet. The deep Web: Surfacing hidden value. White Paper, July 2000.

Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the Web. *Computer Networks*, 29(8-13):1157–1166, 1997.

Soumen Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, San Fransisco, USA, 2003.

Bibliography II

Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.

Kevin Chen-Chuan Chang, Bin He, Chengkai Li, Mitesh Patel, and Zhen Zhang. Structured databases on the Web: Observations and implications. *SIGMOD Record*, 33(3):61–70, September 2004.

Kevin Chen-Chuan Chang, Bin He, and Zhen Zhang. Toward large scale integration: Building a metaquerier over databases on the Web. In *Proc. CIDR*, Asilomar, USA, January 2005.

Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *Proc. VLDB*, Cairo, Egypt, September 2000.

Bibliography III

Muhammad Faheem and Pierre Senellart. Demonstrating intelligent crawling and archiving of web applications. In *Proc. CIKM*, pages 2481–2484, San Francisco, USA, October 2013a. Demonstration.

Muhammad Faheem and Pierre Senellart. Intelligent and adaptive crawling of Web applications for Web archiving. In *Proc. ICWE*, pages 306–322, Aalborg, Denmark, July 2013b.

Muhammad Faheem and Pierre Senellart. Adaptive crawling driven by structure-based link classification, July 2014. Preprint available at `http://pierre.senellart.com/publications/faheem2015adaptive.pdf`.

Georges Gouriten and Pierre Senellart. API Blender: A uniform interface to social platform APIs. In *Proc. WWW*, Lyon, France, April 2012. Developer track.

Bibliography IV

Georges Gouriten, Silviu Maniu, and Pierre Senellart. Scalable, generic, and adaptive systems for focused crawling. In *Proc. Hypertext*, Santiago, Chile, September 2014. Douglas Engelbart Best Paper Award.

IETF. Request For Comments 791. Internet Protocol. `http://www.ietf.org/rfc/rfc0791.txt`, September 1981a.

IETF. Request For Comments 793. Transmission Control Protocol. `http://www.ietf.org/rfc/rfc0793.txt`, September 1981b.

IETF. Request For Comments 1738. Uniform Resource Locators (URLs). `http://www.ietf.org/rfc/rfc1738.txt`, December 1994.

IETF. Request For Comments 1034. Domain names—concepts and facilities. `http://www.ietf.org/rfc/rfc1034.txt`, June 1999a.

Bibliography V

IETF. Request For Comments 2616. Hypertext transfer protocol—HTTP/1.1. `http://www.ietf.org/rfc/rfc2616.txt`, June 1999b.

IETF. Request For Comments 2965. HTTP state management mechanism. `http://www.ietf.org/rfc/rfc2965.txt`, October 2000.

Wallace Koehler. A longitudinal study of web pages continued: a consideration of document persistence. *Inf. Res.*, 9(2), 2003.

Martijn Koster. A standard for robot exclusion. `http://www.robotstxt.org/orig.html`, June 1994.

Jayant Madhavan, Alon Y. Halevy, Shirley Cohen, Xin Dong, Shawn R. Jeffery, David Ko, and Cong Yu. Structured data meets the Web: A few observations. *IEEE Data Engineering Bulletin*, 29(4):19–26, December 2006.

Bibliography VI

Richi Nayak, Pierre Senellart, Fabian M. Suchanek, and Aparna Varde. Discovering interesting information with advances in Web technology. *SIGKDD Explorations*, 14(2), December 2012.

Andrew Sellers, Tim Furche, Georg Gottlob, Giovanni Grasso, and Christian Schallhart. Exploring the Web with OXPath. In *LWDM*, 2011.

Pierre Senellart. Identifying Websites with flow simulation. In *Proc. ICWE*, pages 124–129, Sydney, Australia, July 2005.

Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-Web sources with domain knowledge. In *Proc. WIDM*, pages 9–16, Napa, USA, October 2008.

sitemaps.org. Sitemaps XML format. `http://www.sitemaps.org/protocol.php`, February 2008.

Bibliography VII

Marc Spaniol, Dimitar Denev, Arturas Mazeika, Gerhard Weikum, and Pierre Senellart. Data quality in web archiving. In *Proceedings of the 3rd Workshop on Information Credibility on the Web*, 2009.

W3C. HTML5, October 2014. https://www.w3.org/TR/2014/REC-html5-20141028/.