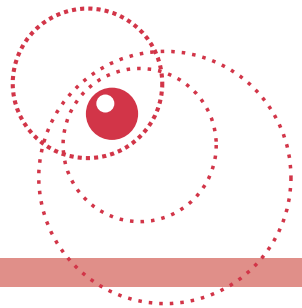


Web Crawling

23 November 2015

TELECOM ParisTech

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

1969 ARPANET (the ancestor of the Internet)

1974 TCP (Vinton G. Cerf & Robert E. Kahn, Turing award winners 2004)

1990 World Wide Web, HTTP, HTML (Tim Berners-Lee, Robert Cailliau)

1993 Mosaic (the first public successful graphical browser, ancestor of Netscape)

1994 Yahoo! (David Filo, Jerry Yang)

1994 Foundation of the W3C

1995 Amazon.com, Ebay

1995 Internet Explorer

1995 AltaVista (Louis Monier, Michael Burrows)

1998 Google (Larry Page, Sergey Brin)

2001 Wikipedia (Jimmy Wales)

2004 Mozilla Firefox

2005 YouTube

Sources: (**?**), (**?**)

`https://www.example.com:443/path/to/doc?name=foo&town=bar#para`

scheme hostname port path query string 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`

- For content: HTML/XHTML, but also PDF, Word documents, text files, XML (RSS, SVG, MathML, etc.)...
- For presenting this content: CSS, XSLT
- For animating this content: JavaScript, AJAX, VBScript...
- For interaction-rich content: Flash, Java, Sliverlight, ActiveX, `<canvas>` API...
- Multimedia content: images, sounds, videos...
- And on the server side: any programming language and database technology to serve this content, e.g., PHP, JSP, Java servlets, ASP, ColdFusion, etc.

Quite complex to manage! Being a Web developer nowadays requires mastering a lot of different technologies; designing a Web client requires being able to handle a lot of different technologies!

- formalized by the W3C (World Wide Web Consortium) formed of industrials (Microsoft, Google, Apple. . .) and academic institutions (ERCIM, MIT, etc.)
- 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.)

- 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.

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)

- 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.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<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.
- The language of the document is specified with the `lang` attribute of the main `<html>` tag.

- 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 files, etc. The two most important items are:
  - The character set of the page, usually at the very beginning of the header

```
<meta http-equiv="Content-Type"
      content="text/html; 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 repertoire, assigning to each character, whatever its script or language, an integer number.

Examples

A	$\rightarrow$	65	ε	$\rightarrow$	949
é	$\rightarrow$	233	א	$\rightarrow$	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.

- `<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.

- 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 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 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.

HTML 4.01 (1999) strict (as described earlier) and transitional

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

- XHTML 1.0 (2000) strict and transitional
- XHTML 1.1 and XHTML 2.0: mostly a failure, unusable and unused in today's Web
- HTML5: latest standard, partly implemented, continuously updated

```
<!DOCTYPE html>
```

- A lot of 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.

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. . .

- XHTML 2.0: initiative of the W3C, incompatible with HTML 4.01/XHTML 1.0, major changes
- HTML5: initiative of browser developers, compatible with HTML 4.01/XHTML 1.0, incremental but numerous changes
- XHTML 2.0 abandoned in July 2009
- HTML5 features have appeared in recent browsers (Internet Explorer 9 included)
- HTML5 offers the choice between syntactic conventions inherited from both HTML 4.01 and XHTML
- New features: 2D drawing (`<canvas>`), multimedia (`<audio>` , `<video>`), better structuring elements (`<section>` , `footer`), etc.

- Application protocol at the basis of the World Wide Web
  - Latest and most widely used version: HTTP/1.1

- 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).

- 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
```

- 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
```

- 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`.

- 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

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

- 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
```

- The browser behaves differently depending on the content type identified: 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
```


- 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
```


- 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
```

- 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
```

- 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/
```


- 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

- 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 (**?**)

Web-scale

- The Web is infinite! Avoid robot traps by putting depth or page number limits on each Web server
- Focus on important pages (**?**)

- Web servers under a list of DNS domains: easy filtering of URLs
- A given topic: focused crawling techniques (**??**) 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? (**?**)
- A given Web site: what is a Web site? (**?**)

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.

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!

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 the edit distance 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$\}$

- 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)

- Standard for robot exclusion: robots.txt at the root of a Web server (**?**).

```
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

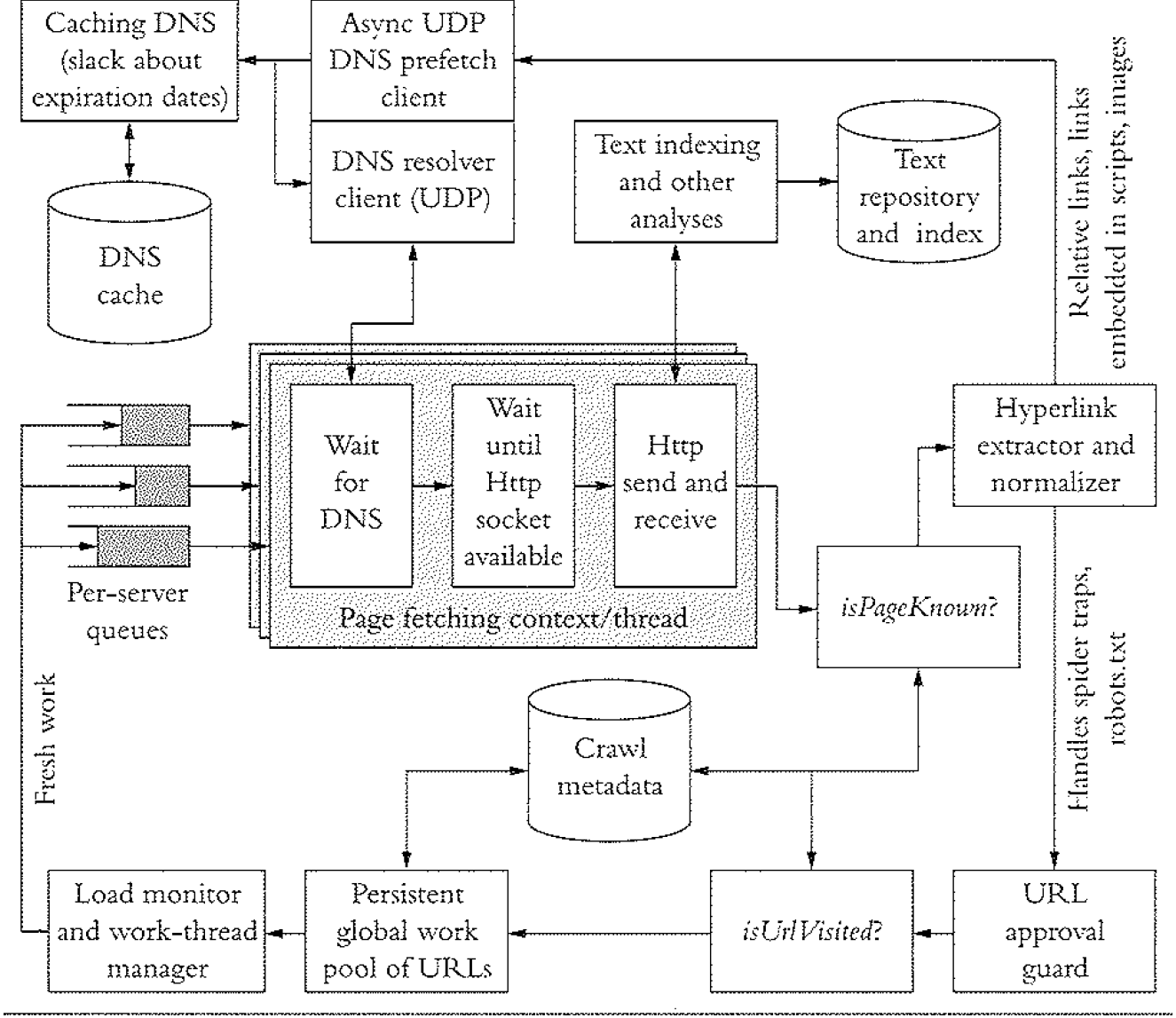
- General principles:
  - t access or keep access to a “system for automated data processing” *in a fraudulous manner* is punished of two years of prison and 60,000 euros fine (Code pénal 323-1, modified by law 2015-912 on “Renseignement”)
  - to disrupt the functioning of a “system for automated data 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”)

robots.txt files can be taken are 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 (Loi "Informatique et Libertés"): personal data should be collected for a specific purpose, and should be kept updated

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


Caching DNS (slack about expiration dates)
Async UDP DNS prefetch client
DNS resolver client (UDP)
DNS cache
Text indexing and other analyses
Text repository and index
Relative links, links embedded in scripts, images
Wait for DNS
Wait until Http socket available
Http send and receive
Page fetching context/thread
Per-server queues
Hyperlink extractor and normalizer
isPageKnown?
Handles spider traps, robots.txt
Fresh work
Crawl metadata
Load monitor and work-thread manager
Persistent global work pool of URLs
isUrlVisited?
URL approval guard

- 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

- 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

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, Internet Explorer)

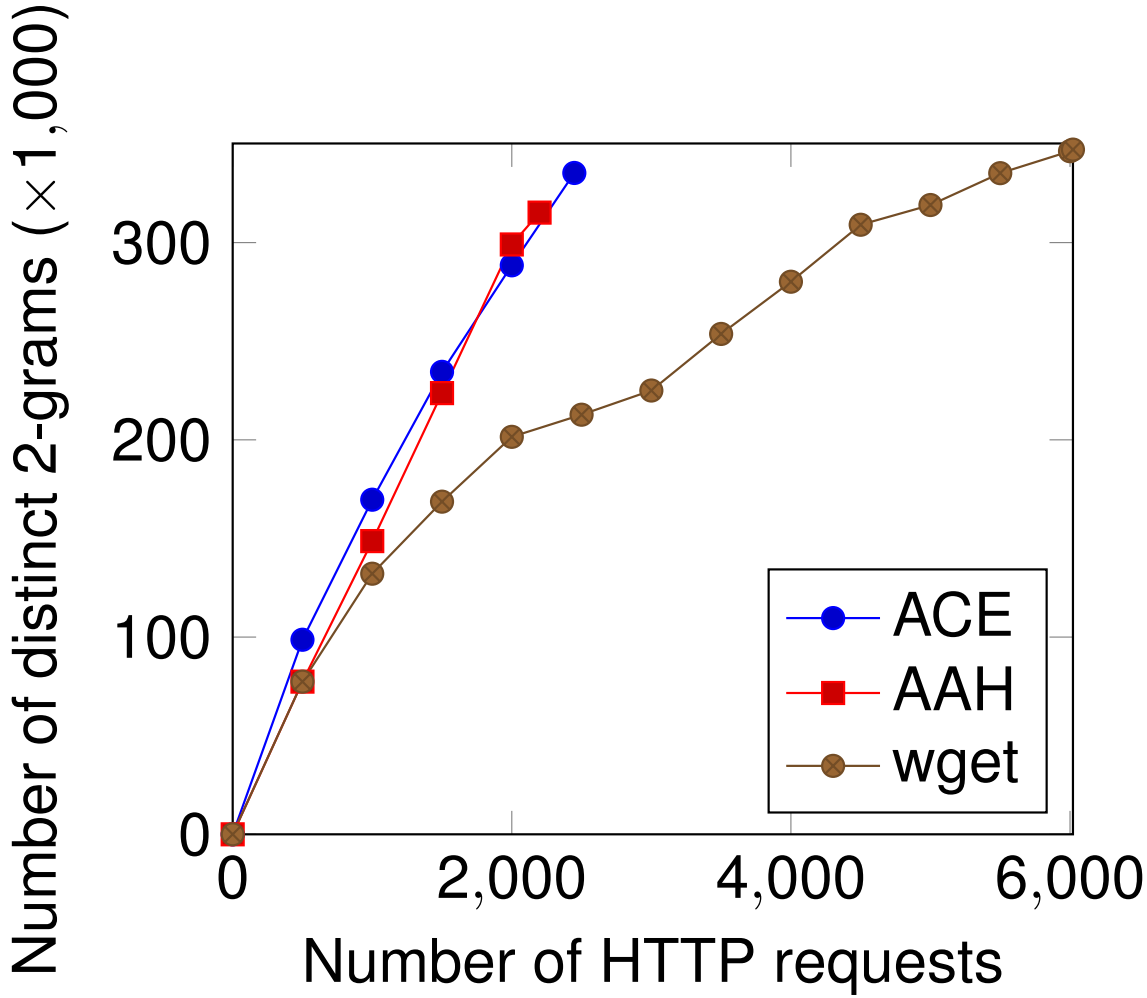
OXPath: a full-fledged navigation and extraction language for complex Web sites (**?**)

TELECOM ParisTech

- 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

- 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 (**??**) (AAH)
  - Automatically infer the best way to crawl a given CMS (**?**) (ACE)
- Need to be robust w.r.t. template change

- 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.



Huge numbers of users (2012):

Facebook	900 million
QQ	540 million
W. Live	330 million
Weibo	310 million
Google+	170 million
Twitter	140 million
LinkedIn	100 million

Huge numbers of users (2012):

Facebook	900 million
QQ	540 million
W. Live	330 million
Weibo	310 million
Google+	170 million
Twitter	140 million
LinkedIn	100 million

Huge volume of shared data:

250 million tweets per day on Twitter (3,000 per second on average!)...

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

- Theoretically possible to crawl social networking sites using a regular Web crawler
- Sometimes not possible:
  `https://www.facebook.com/robots.txt`
- Often very inefficient, considering politeness constraints
- Better solution: Use provided social networking APIs
  `https://dev.twitter.com/docs/api/1.1`
  `https://developers.facebook.com/docs/graph-api/reference/v2.1/`
  `https://developer.linkedin.com/apis`
  `https://developers.google.com/youtube/v3/`
- Also possible to buy access to the data, directly from the social network or from brokers such as `http://gnip.com/`

- 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 require 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://dev.twitter.com/docs/api/1.1/get/search/tweets`

- 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

- Two main APIs:
  - REST APIs, including search, getting information about a user, a list, followers, etc. `https://dev.twitter.com/docs/api/1.1`
  - Streaming API, providing real-time result
- Very limited history available
- Search can be on keywords, language, geolocation (for a small portion of tweets)

- Often useful to combine results from different social networks
- Numerous libraries facilitating SN API accesses (twipy, Facebook4J, FourSquare VP C++ API...) incompatible with each other... Some efforts at generic APIs (OneAll, APIBlender (**?**))
- 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.

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
- Ideally: a traditional large-scale crawler that knows when to delegate to more specialized crawling mechanisms (tools querying social networking APIs, JS-aware crawlers, etc.)
- Huge variety of tools, techniques, suitable for different needs

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 (**?**)
- Main references:
  - HTML 4.01 recommendation (**?**)
  - HTTP/1.1 RFC (**?**)