Web Crawling

- Najork and Heydon, *High-Performance Web Crawling*, Compaq SRC Research Report 173, 2001. Also in Handbook of Massive Data Sets, Kluwer, 2001.
- Najork and Wiener, *Breadth-first search crawling yields high-quality pages*. Proc. 10th Int. WWW Conf., 2001.

Web Crawling

Web Crawling = Graph Traversal

S = {startpages}
repeat
    remove an element s from S
    foreach (s, v)
        if v not crawled before
            insert v in S

Issues

Theoretical:

- Startset S
- Choice of s (crawl strategy)
- Refreshing of changing pages.

Practical:

- Load balancing (own resources and resources of crawled sites)
- Size of data (compact representations)
- Performance (I/Os).

Crawl Strategy

- Breath First Search
- Depth First Search
- Random
- Priority Search

Possible priorities:

- Often changing pages (how to estimate change rate?).
- Using global ranking scheme for queries (e.g. PageRank).
- Using query dependent ranking scheme for queries (“focused crawling”, “collection building”).

BFS is Good

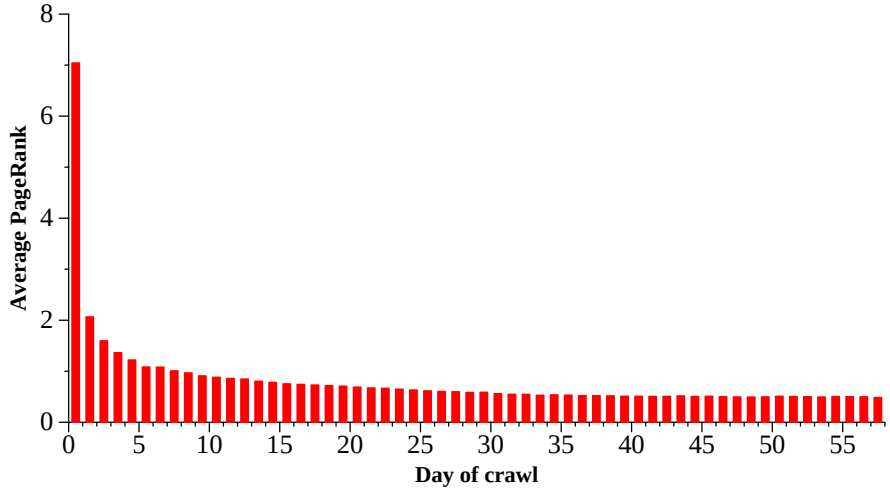


Figure 1: Average PageRank score by day of crawl

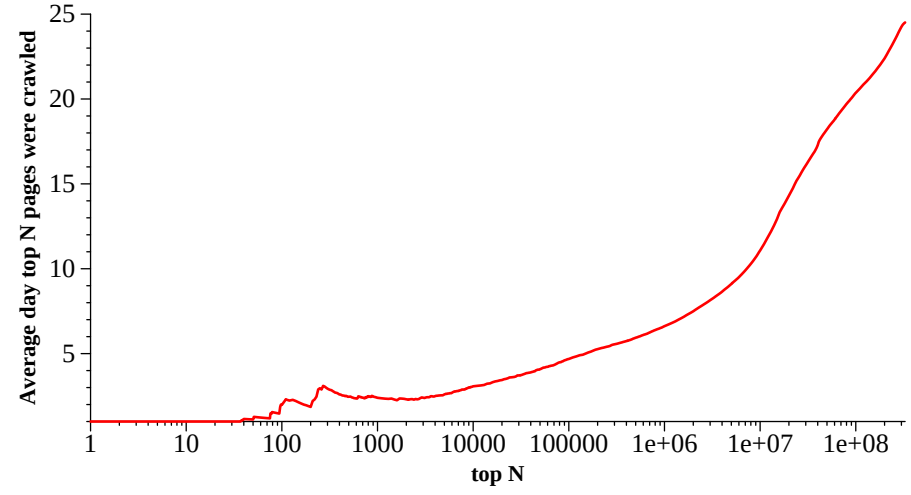


Figure 2: Average day on which the top N pages were crawled

[From: Najork and Wiener, 2001]

Statistics for crawl of 328 million pages.

PageRank Priority is Even Better

(but computationally expensive to use. . .)

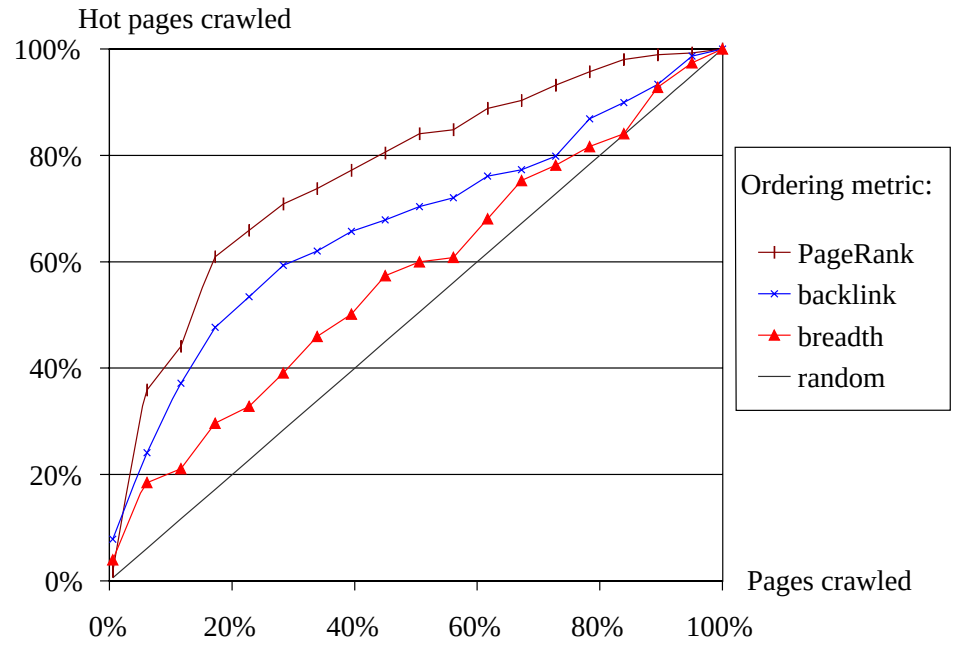


Figure 2: The performance of various ordering metrics for $IB(P)$; $G = 100$

[From: Arasu et al., *Searching the Web*. ACM Trans. Internet Technology, 1, 2001]

Statistics for crawl of 225.000 pages at Stanford.

Load Balancing

Own resources:

- Bandwidth (control global rate of requests)
- Storage (compact representations, compression)
- Industrial-strength crawlers must be distributed (e.g. partition the url-space)

Load Balancing

Own resources:

- Bandwidth (control global rate of requests)
- Storage (compact representations, compression)
- Industrial-strength crawlers must be distributed (e.g. partition the url-space)

Resources of others:

- BANDWIDTH. Control local rate of requests (e.g. 30 sec. between request to same site).
- Identify yourself in request. Give contact info (mail and www).
- Monitor the crawl.
- Obey the Robots Exclusion Protocol (see `www.robotstxt.org`).

Efficiency

- RAM: never enough for serious crawls. Efficient use of disk based storage important. I/O when accessing data structures is often a bottleneck.
- CPU cycles: not a problem (Java and scripting languages are fine).
- DNS lookup can be a bottleneck if using synchronized version. Brug asynchronous DNS (e.g. GNU `adns` library).

Rates reported for serious crawlers: 200-400 pages/sec.

Crawler Example: Mercator

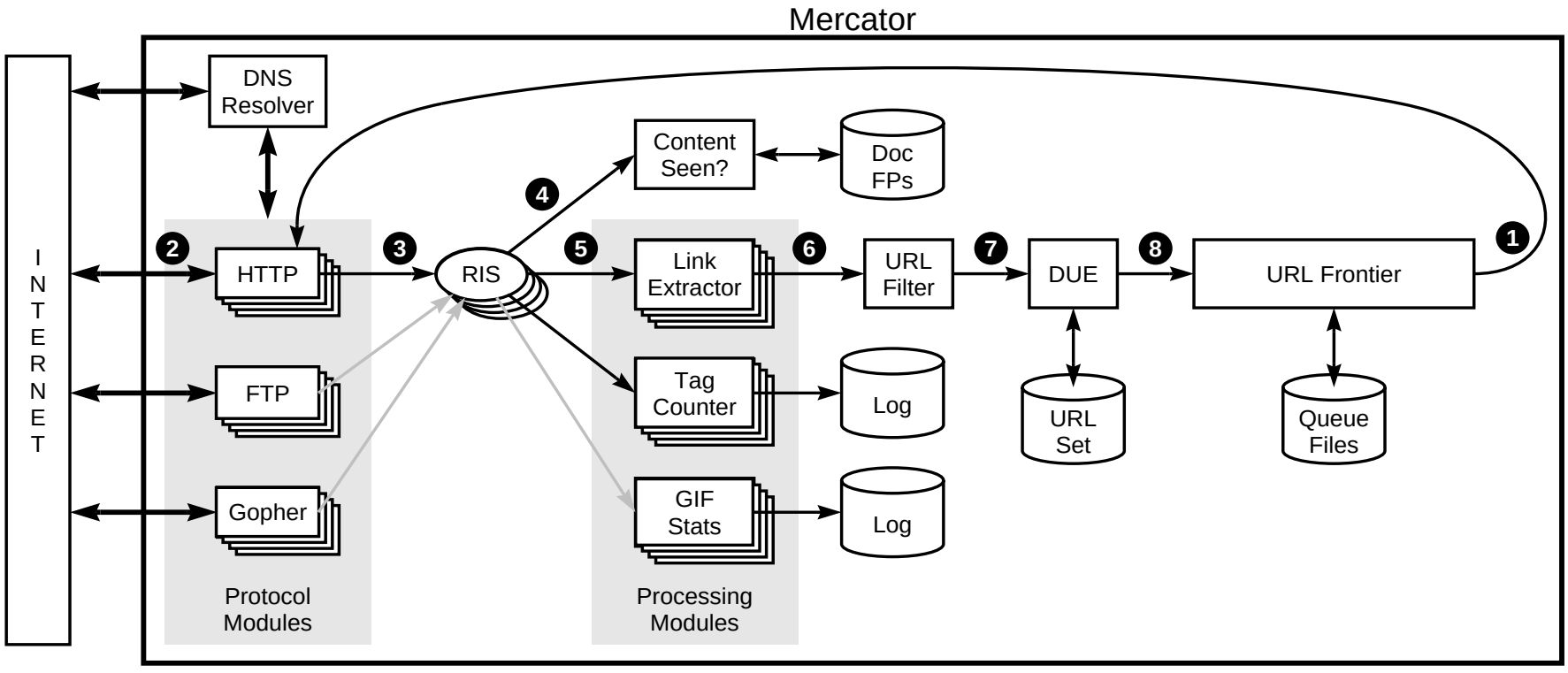


Figure 1: Mercator's main components.

[From: Najork and Heydon, 2001]

Mercator

Further features:

- Uses fingerprinting ((sparse) hashfunction on strings) for URL IDs (see e.g. ex. `md5` (128 bit) or the `sha` family (160-512 bits)).
- Continuous crawling—crawled pages put back in queue (prioritized using update history).
- Checkpointing (crash recovery).
- Very modular structure.

Details: Politeness

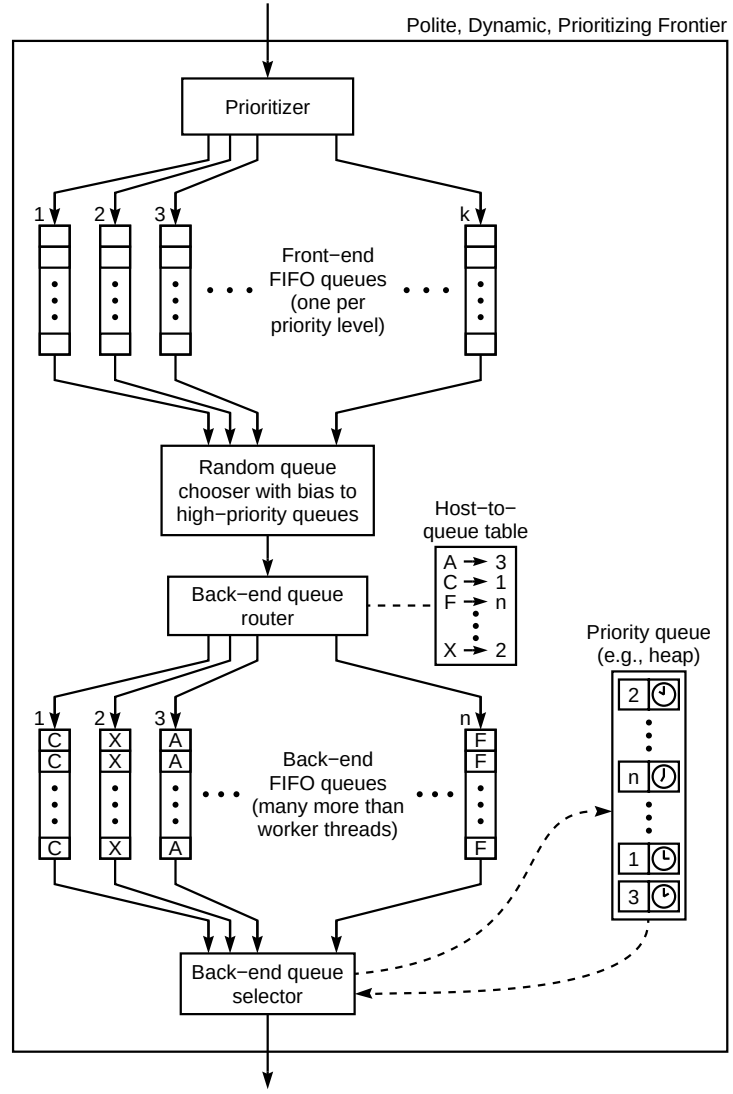


Figure 3: Our best URL frontier implementation

[From: Najork and Heydon, 2001]

Details: Efficient URL Elimination

- Fingerprinting
- Sorted file of fingerprints of seen URLs.
- Cache most used URLs.
- Non-cached URLs checked in batches (merge with file I/O).

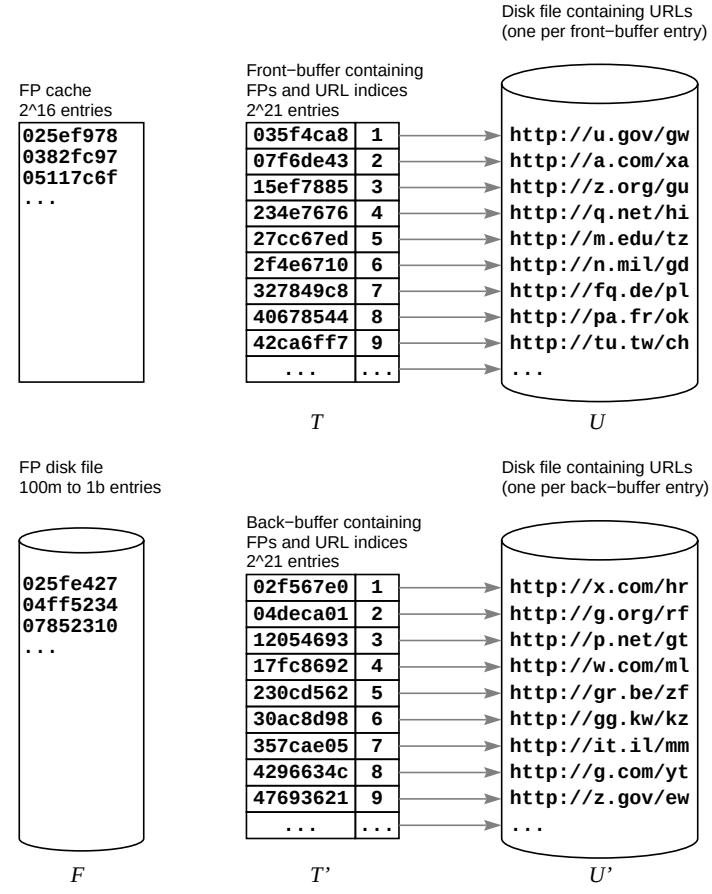


Figure 4: Our most efficient disk-based DUE implementation

[From: Najork and Heydon, 2001]

Details: Parallelization

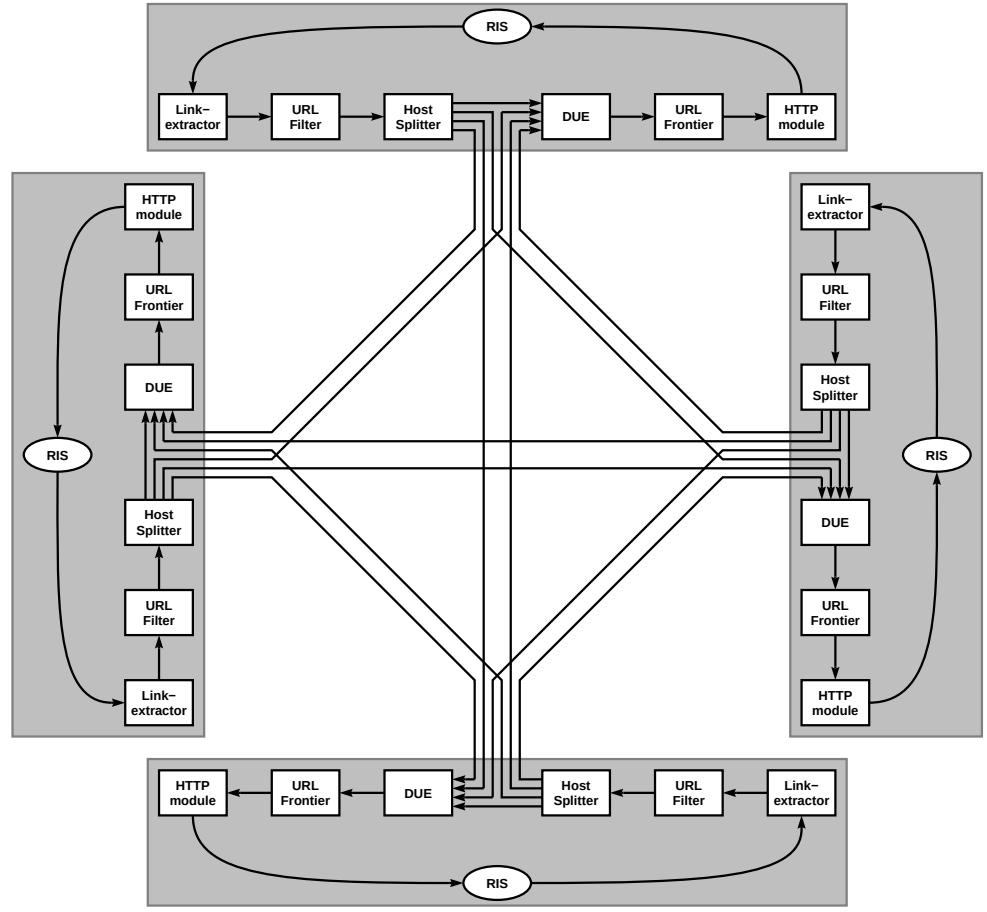


Figure 2: A four-node distributed crawling hive

[From: Najork and Heydon, 2001]

Some Experiences

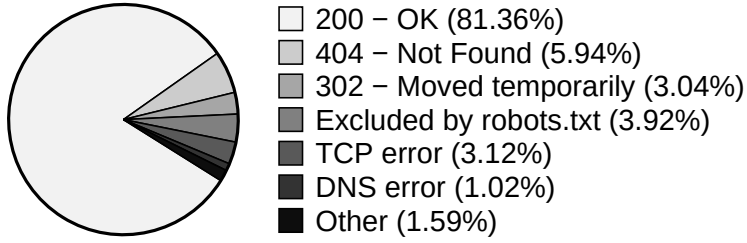


Figure 6: Outcome of download attempts

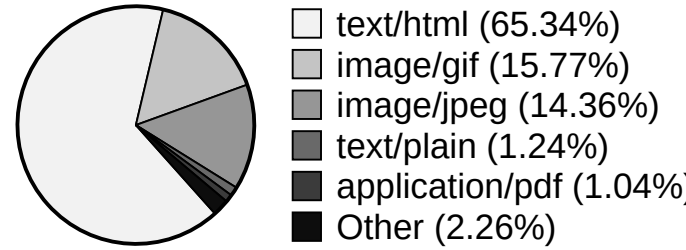


Figure 7: Distribution of content types

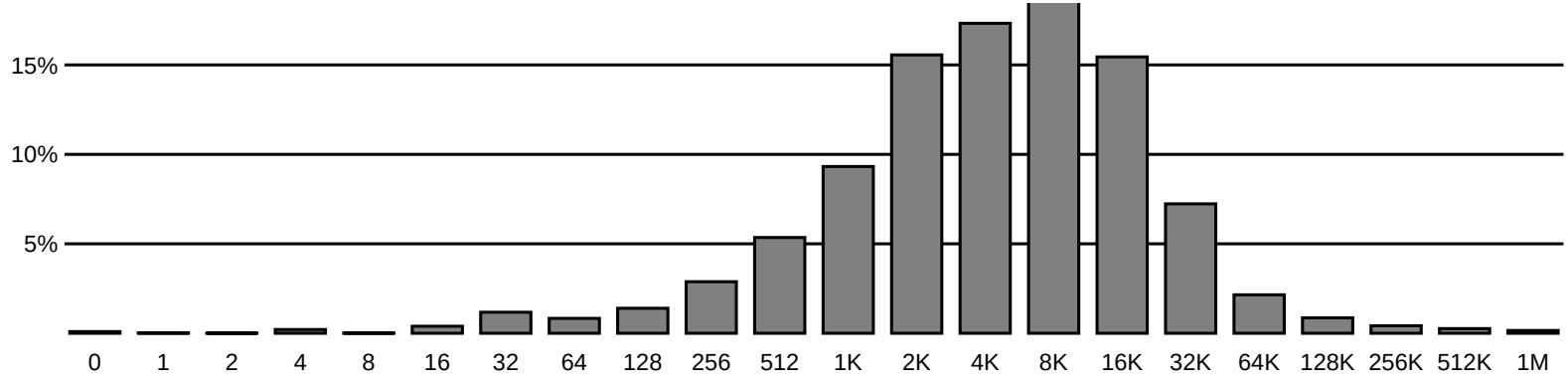
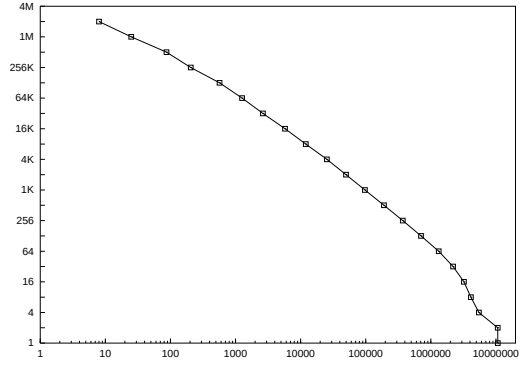
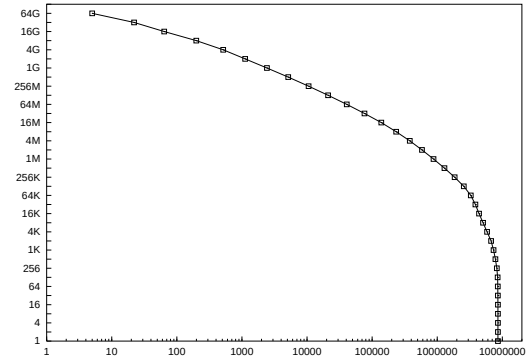


Figure 8: Distribution of document sizes

Some Experiences

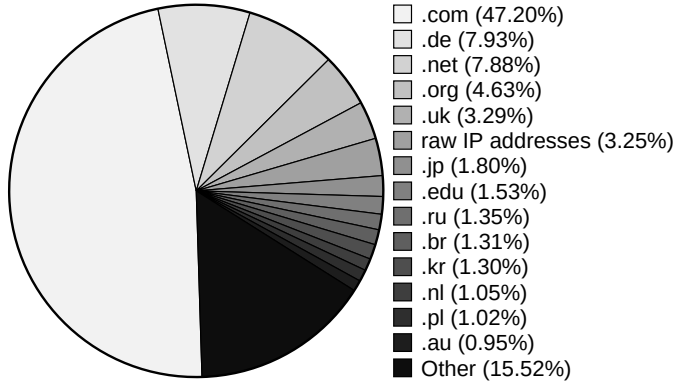


(a) Distribution of pages over web servers

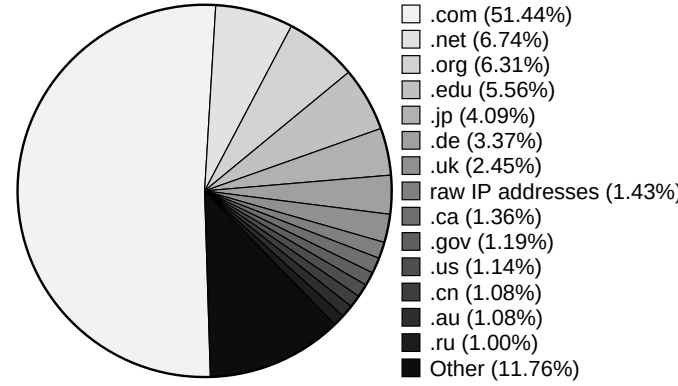


(b) Distribution of bytes over web servers

Figure 9: Document and web server size distributions



(a) Distribution of hosts over



(b) Distribution of pages over

[From: Najork and Heydon, 2001]

Robot Exclusion Protocol

Simple protocol suggested by Martijn Koster in 1993. De facto standard for robot exclusion. Full details at `www.robotstxt.org`.

- Single file named `robots.txt` in root of server.
- Contains simple directions for exclusion of parts of site.

Example:

```
User-agent:  *
Disallow:  /cgi-bin/
Disallow:  /tmp/
Disallow:  /joe/

User-agent:  BadBot
Disallow:  /
```

Robot Exclusion in HTML

Per page exclusion through the `META` tag in HTML.

Example:

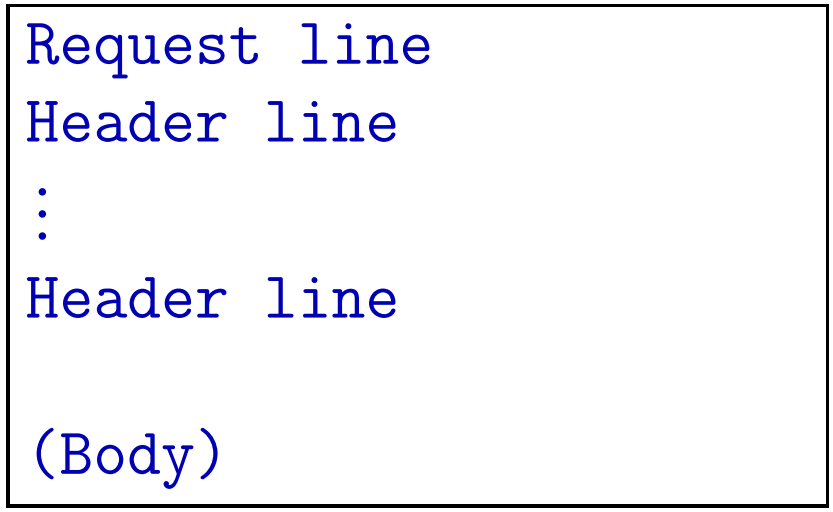
```
<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
```

Further details at `www.w3.org/TR/html4/` (the HTML 4.01 specification) and at `www.robotstxt.org`

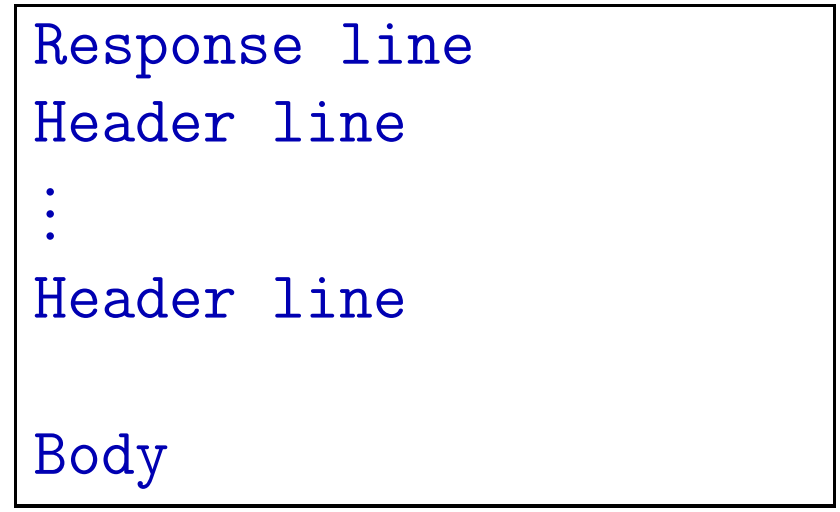
HTTP Protocol

One request message, one response message (over a single TCP connection).

Format of messages:



Request



Response

HTTP Example

```
GET /somedir/page.html HTTP/1.1
Host:  www.somefirm.com
Accept:  text/*
User-Agent:  Mozilla 7.0 [en]
```

Request

```
HTTP/1.1 200 OK
Content-Type:  text/html
Content-Length:  345

<HTML>
<HEAD>
⋮
```

Response

URLs

Absolute:

`http://www.somefirm.dk:80/main/test`
`http://www.somefirm.dk/main/test#thirdEntry`
`http://www.somefirm.dk/cgi-bin?item=123`

Relative:

`./dir/test.html`

Relative to

- URL of doc containing URL
- URL specified in `<BASE>` HTML tag.

Encoded characters:

`www.sdu.dk/~rolf` $\longrightarrow$ `www.sdu.dk/%7Erolf`

Normalizing URLs

- Add portnumber if not present (`:80`).
- Convert escaped chars to real chars.
- Remove `...#target` from URL.