HTTP continued...

Claudia Hauff
TI1506: Web and Database Technology
ti1506-ewi@tudelft.nl

TU Delft

Uniform Resource Locators (URLs)

URL syntax

- Uniform resource locators offer a standardised way to point to any resource on the Internet

- Not restricted to the `http` scheme, syntax slightly varies from scheme to scheme

- General format (adhered to by most schemes):

```
<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>
```

URL syntax

<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>

The name of a piece of a resource. Only used by the client - the fragment is not transmitted to the server.

Parameters passed to gateway resources, i.e. applications [identified by the path] such as search engines.

Additional input parameters applications may require to access a resource on the server correctly. Can be set per path segment.

the local path to the resource

the port on which the server is expecting requests for the resource

domain name (host name) or numeric IP address of the server

the username/password (may be necessary to access a resource)

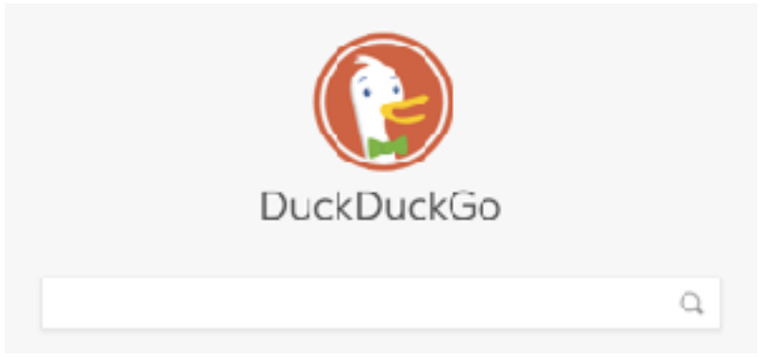
determines the protocol to use when connecting to the server.

URL syntax: query
(most important to us)

<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>

https://duckduckgo.com/html?q=delft

- Query component is passed to the application accessed at the Web server (“gateway resource”)
- Necessary to enable interactive applications (like our habit tracker Web application!)
- Common convention: **`name1=value1&name2=value2&…`**



URL design restrictions

- **Initial design goals**: *portable* across protocols and human *readable* (no invisible/non-printing chars.)
- URLs initially restricted to a very small “safe” alphabet: ASCII **Heavily biased in favor of English speakers**

  latin alphabet, `0123456789 - _ .~`
  and additional reserved chars like `! ( ) @ &`

- **Added later**: character encoding, e.g. whitespace as `%20`

`http://правительство.рф`

Punycode (RFC3492)

"Punycode is a simple and efficient transfer encoding syntax designed for use with Internationalized Domain Names in Applications. It **uniquely** and **reversibly** transforms a Unicode string into an ASCII string."

```
http://правительство.рф
      => http//:xn--80aealotwbjpid2k.xn--p1ai
http://nl.wikipedia.org/wiki/Itali%C3%AB
      => http://nl.wikipedia.org/wiki/Itali%C3%AB
```

A potential security issue in *mixed scripts*: http://paypal.com

IDNA demo: http://demo.icu-project.org/icu-bin/idnbrowser

Authentication

Authentication: a system verifying the identity of a user who requests access.

Authentication

So far: HTTP as **anonymous**, **stateless** request/response protocol. The same request, sent by different clients, is treated in exactly the same manner.

Now: identification via

A. **HTTP headers**
B. **Client IP address tracking**
C. **Fat URLs**
D. **User login** (HTTP Basic Authentication)

In later lectures: Cookies & Sessions

User-related HTTP header fields

From	Request	User's email address	**mostly Web crawler**
User-Agent	Request	User's browser	**device customization**
Referer	Request	Page the user came from	**user interests**
Client-IP	Request (Extension)	Client's IP address	
Authorization	Request	Username & password	

Client IP address tracking

- **Idea**: **Client IP address** as user identifier (if not in HTTP header, then via the TCP connection)
- **Many problems**:
  A. IP addresses describe the **machine**, not the user
  B. Internet service providers **dynamically assigned IP** addresses to users
  C. Users may access the Web through **firewalls** (obscures the real IP address)
  D. HTTP **proxies** and **gateways** open new TCP connections (IP of the proxy/gateway is shown), **`X-Forwarded-For`** might help (it quickly gets complicated)

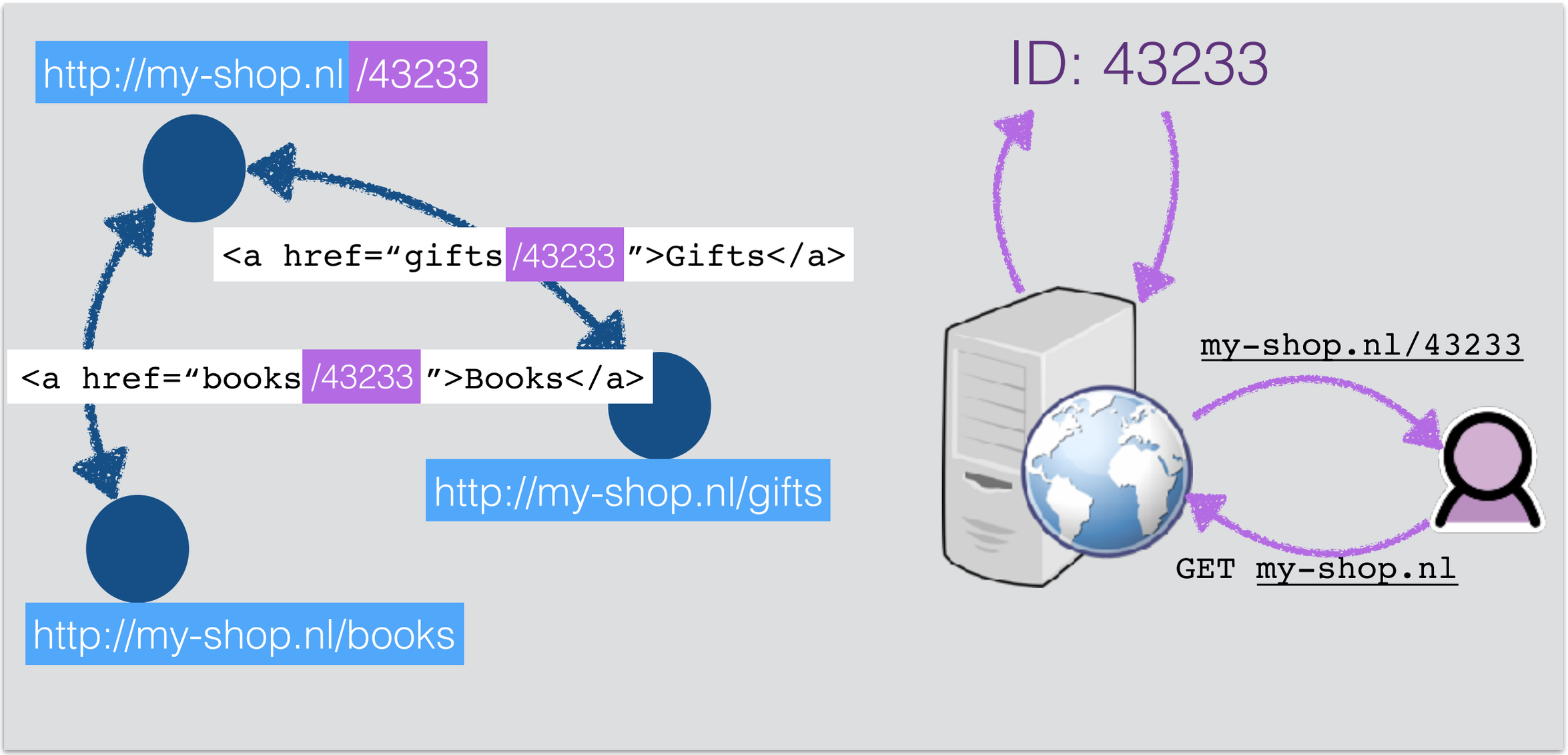
Fat URLs

- Tracking through the **generation of unique URLs** for each user
  - First time a user visits a page within a Web site, a **unique ID** is generated by the server
  - Server **redirects** client to the fat URL
  - Server on the fly **rewrites the HTML** when an HTTP request with a fat URL is received (adds ID to all hyperlinks to maintain the knowledge)

```
<a href="/browse/002-1145265-8016838">Gifts</a>
<a href="/wishlist/002-1145265-8016838">Wish List</a>
```

- Independent HTTP transactions can be tied into a single "session"

Fat URLs


http://my-shop.nl /43233
Gifts
Books
http://my-shop.nl/gifts
http://my-shop.nl/books
ID: 43233
my-shop.nl/43233
GET my-shop.nl

Fat URLs

- Fat URLs are **ugly**
- Fat URLs **cannot be shared** (URL may not exist later or user inadvertently shares private info)
- Fat URLs **break caching** mechanisms (one URL per user/page instead of one URL per page)
- **Extra server load** (HTML page rewrites necessary)
- Users can "**escape**" (ID is lost when user navigates outside the Web site)

HTTP basic authentication

- Server explicitly asks the user for authentication (**username and password**)

- HTTP has a **built-in mechanism** to support username/password based authentication via `WWW-Authenticate` and `Authorization` headers

- HTTP is **stateless**: once logged in, the client sends the login information with each request

HTTP basic authentication

Put into practice in Assignment 1

1

```
GET /index.html HTTP/1.1
host: www.microsoft.com
```

```
HTTP/1.1 401 Login Required
WWW-Authenticate: Basic realm="B&R"
```

2

security realm and authentication algorithm

client presents login screen

3

```
GET /index.html HTTP/1.1
host: www.microsoft.com
Authorization: Basic am910jRmdW4=
```

4

```
HTTP/1.1 200 OK
Content-length: 1234
Content-type: text/html
```

In future HTTP requests to the site, the browser automatically issues the stored username/password.

HTTP basic authentication

`Authorization: Basic`

- Username and password are joined together by a colon and converted to **base-64 encoding** (binary-to-text encoding)

- Base-64 encoding ensures that **only HTTP compatible characters** are entered into the message (takes as input binary, text and international character data strings)

`Normandië`	`Tm9ybWFuZGnDqw==`
`Delft`	`RGVsZnQ=`
`España`	`RXNwYcOxYQ==`

Images can be encoded this way too!

HTTP basic authentication: secure?

- Username and password can be **decoded trivially** (sent over the network “in the clear”)

- Users tend to **reuse** login/password combinations; a non-critical web site may use basic authentication without SSL that an opponent can capture and try on critical sites

HTTP basic authentication: overall

Basic authentication prevents **accidental** or **casual access** by curious users (privacy is desired but not essential).

Basic authentication is useful for **personalisation** and access control within a “friendly” environment (intranet).

“In the wild”, basic authentication should always be used in combination with **secure HTTP (e.g. https)** — avoids sending username/password **in the clear** across the network.

Secure HTTP

Secure HTTP

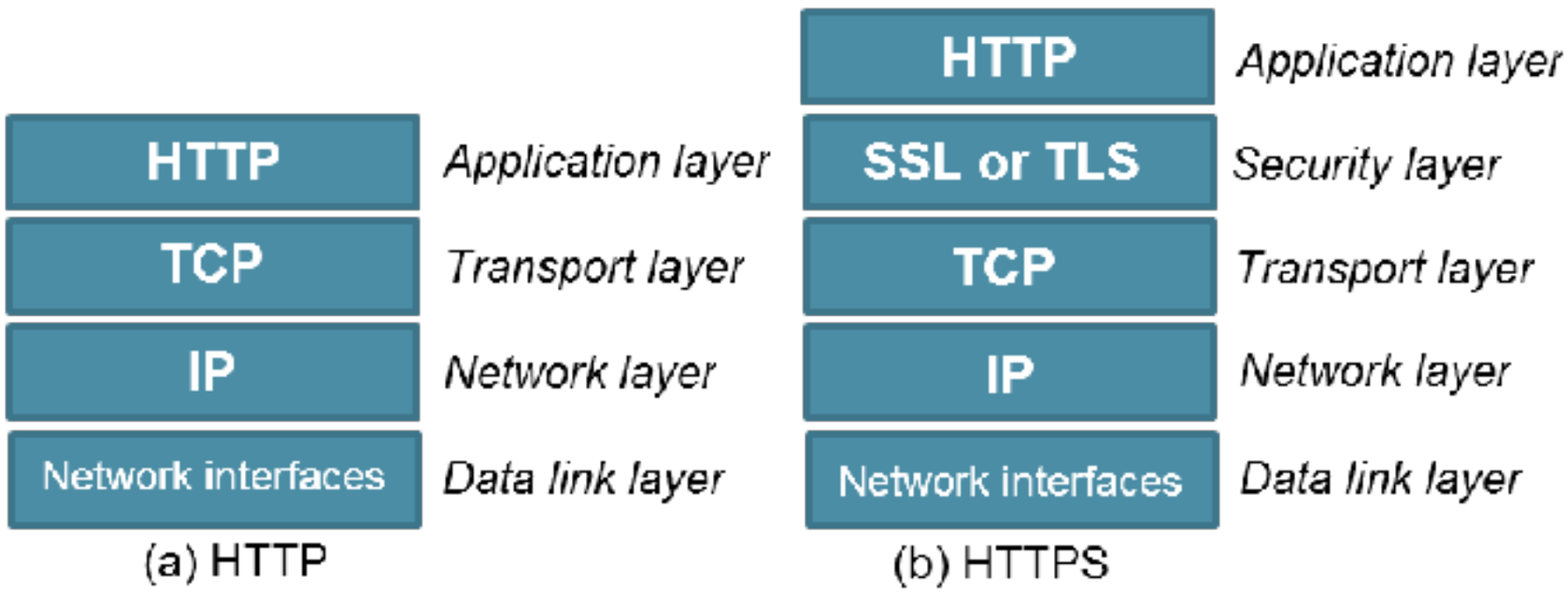
- So far: **lightweight** authentication
  - Not useful for purchasing, bank transactions or confidential data
- Secure HTTP should provide:

A. **Server authentication** (client is sure to talk to the right server)
B. **Client authentication** (server is sure to talk to the right client)
C. **Integrity** (client and server are sure their data is intact)
D. **Encryption**
E. **Efficiency**

Secure HTTP: HTTPS

- HTTPS is the most popular secure form of HTTP
- URL scheme is **`https://`** instead of **`http://`**

- Request and response data are **encrypted** before being sent across the network (SSL: Secure Socket Layer)



Client & server **negotiate** the cryptographic protocol to use.