PSI
Programowanie Sieciowe
zima 2023-2024

Grzegorz Blinowski

Instytut Informatyki

Politechniki Warszawskiej

Protokół HTTP

HTTP/1.1 – oryg. 2616
HTTP/1.1 – **RFC 7230,7231**
+ szereg uzupełnień
oraz: RFC 2817

HTTP - protokół bezstanowy

w przeciwieństwie do FTP i SMTP, HTTP nie wymaga przechowania informacji o stanie sesji przez strony komunikujące się (łatwiejszy w implementacji, bardziej niezawodny)

Uproszona sesja HTTP (z przeglądarki WWW):

U: *"wpisuje" adres:* `http://www.ii.pw.edu.pl`

K: *DNS lookup, określenie IP*

K: wysłanie "GET / HTTP/1.1\cr\lf\cr\lf" do serwera

S: decyduje jak obsłużyć żądanie

S: wysyła nagłówek odpowiedzi i opcjonalnie dane

K, S: sesja TCP jest zamykana

HTTP - przykład

```
% telnet www.ii.pw.edu.pl 80
Trying...
Connected to www.ii.pw.edu.pl.
Escape character is '^]'.
GET / HTTP/1.1


HTTP/1.1 200 OK
Date: Mon, 10 May 2021 19:07:09 GMT
Server: Apache/2.2.15 (Red Hat)
Content-language: pl-PL
...
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl-PL"
   lang="pl-PL"><head>
```

HTTP - podstawy

Żądanie (request)

```
GET /home.html HTTP/1.1
User-agent: Mozilla/5.0
Host: www.ii.pw.edu.pl
Accept-language:en

(CR. LF)
```

żądanie

nagłówek

Odpowiedź (response)

```
HTTP/1.1 200 OK
Date: Sat, 1 May 2021 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Thu, 1 Apr 2021 ...
Content-Length: 6166
Content-Type: text/html

dane dane dane i więcej danych
```

status

nagłówek

Więcej o HTTP

Full-Request = Request-Line
*(General-Header | Request-Header | Entity-Header)
CRLF
[Entity-Body]

Full-Response = Status-Line *(General-Header | Response-Header | Entity-Header)
CRLF
[Entity-Body]

```
GET /cgi-bin/q HTTP/1.1
Connection: Keep-Alive
User-agent: Mozilla/5.0
Host: www.blahblah.com.pl
Accept: text/html
Accept-language:en
ContentType: application/x-www-
   form-urlencoded
```

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Server: Apache/1.3.0 Unix
Last-Modified: Thu, 1 Apr
   2021 ...
Content-Length: 6166
Content-Type: text/html
```

URL, URI, URN

RFC1736, 1737
RFC2396

- URI (Uniform Resource Identifier)
  - **Uniform** - *" it allows different types of resource identifiers to be used in the same context, even when the mechanisms used to access those resources may differ"*
  - **Resource** - *"A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service"*
  - **Identifier** - *"An identifier is an object that can act as a reference to something that has identity. In the case of URI, the object is a sequence of characters with a restricted syntax"*
- URL (Uniform Resource Locator) - podzbiór URI

– określa zasoby poprzez sposób dostępu, np. **http**

- URN (Uniform Resource Name) - podzbiór URI - trwała i unikalna nazwa niezależna od lokalizacji (**isbn:**)

URI, URL - Przykłady

```
ftp://ftp.is.co.za/rfc/rfc1808.txt

http://www.math.uio.no/faq/compression-faq/part1.html

mailto:mduerst@ifi.unizh.ch

news:comp.infosystems.www.servers.unix

telnet://melvyl.ucop.edu/

pop://rg;AUTH=+APOP@mail.eudora.com:8110

ldap://ldap.umich.edu/o=University%20of%20Michigan,c=US

imap://michael@minbari.org/users.*;type=list
```

URL, URI

RFC1738, RFC2616,
https: RFC7230

URL - URI w domenie HTTP, opisane w ABNF:

(notacja nieco uproszczona w stosunku do RFC1738)

```
http_URL = "http:" "//" [login] host [ ":" port ]
    [ abs_path ]

login = user [ ":" password ] "@"

abs_path = "/" rel_path
rel_path = [path] [";" params] ["?"query ]
params = param *( ";" param )
param = *( pchar | "/" )
pchar = uchar | ":" | "@" | "&" | "=" | "+"
```

Składnia URL

Przykład URL:

```
http://www.xyz.com:8888/cgi-bin/a?p1=a11,2&p2=b21#31
```

Znaki "bezpieczne":

```
A-Za-z0-9 "$" | "-" | "_" | "." "!" | "*" |
"'" | "(" | ")" | ","
```

Znaki specjalne (separatory):

```
";" | "/" | "?" | ":" | "@" | "&" | "=" |
"+"
```

Escape - dowolny znak co do którego mamy wątpliwości może zostać zakodowany szesnastkowo (np. separator, " " lub nie ASCII)

```
escape          = "%" HEX HEX
```

URL-e c.d.

Gdzie tu jest adres?

http://www.citibank.com:ac=piUq3027qcHw003nfuJ2@sd96V.pIsEm.NeT/3/?3X6CMW2I2uPOVQW

Odp:

...

URL-e c.d.

Gdzie tu jest adres?

http://
www.citibank.com:ac=piUq3027qcHw003nfuJ2@sd96V.pIsE
m.NeT/3/?3X6CMW2I2uPOVQW

Odp:

http://
www.citibank.com:ac=piUq3027qcHw003nfuJ2@sd96V.pIsE
m.NeT/3/?3X6CMW2I2uPOVQW

Polecenia HTTP (method)

GET - pobierz dane

HEAD - jak GET, ale zwracany tylko nagłówek

POST - zgłoś dane do przetworzenia przez URL

PUT - zmień zawartość URL

DELETE - usuń dane

TRACE - zwróć otrzymane żądanie jako entity

OPTIONS - zwróć dostępne opcje

CONNECT - przełącz proxy w inny tryb (tunel TLS)

WEBDav

(RFC2518)

PUT,

POST,

PROPPATCH,

PROPFIND,

LOCK,

UNLOCK,

MOVE,

MKCOL

Kody odpowiedzi
(status code, reason phrase)

- **1xx**: ***Informational*** - rzadko używane (zmiana protokołu)
- **2xx**: ***Success*** - polecenie zrozumiane, wykonane poprawnie:
  - **200** - OK, **201** - Created, **202 -** Accepted,
  - **204 -** No content (np. ze zgłoszenia formularza), **206 -** Partial Content (w odpowiedzi na "Range")
- **3xx**: ***Redirection*** - należy podjąć dodatkowe czynności w celu realizacji polecenia - patrz Entity - "Location":
  - 300 – Multiple Choices **
  - 301 – Moved Permanently *
  - 302 – Found *
  - 303 – See Other *
  - 304 – Not Modified (w odp. na **If-Modified-Since** lub **If-None-Match**),
  - 305, 306 – nie używane
  - 307 – Temporary redirect *

* w nagłówku **Location** podane gdzie jest dokument

** w nagłówku **Location** może być podane gdzie jest dokument

Kody odpowiedzi
(status code, reason phrase)

- **4xx**: ***Client Error*** - błąd polecenia, prawdopodobie błąd składni po stronie klienta
  - 400 - Bad request,
  - 401 - Unauthorised (dołączone pole **WWW-Authenticate**)
  - 403- Forbiden,
  - 404 - Not found
  - 407 - Proxy Authentication Required
  - 408 - 417 - inne błędy
- **5xx**: ***Server Error*** - serwer nie jest w stanie obsłużyć polecenia
  - 500 - Internal Server Error, 501 - Not Implemented, 503 - Service Unavailable, 505 - HTTP Version Not Supported

Nagłówki HTTP (powtórka)

Full-Request = Request-Line
*(General-Header |
Request-Header |
Entity-Header)
CRLF
[Entity-Body]

Full-Response = Status-Line *(General-Header | Response-Header | Entity-Header)
CRLF
[Entity-Body]

```
GET /cgi-bin/q HTTP/1.1
Connection: Keep-Alive
User-agent: Mozilla/5.0
Host: www.blahblah.com.pl
Accept: text/html
Accept-language:en
ContentType: application/x-www-
   form-urlencoded
```

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Server: Apache/1.3.0 Unix
Last-Modified: Mon, 22 Jun
   2021 ...
Content-Length: 6166
Content-Type: text/html
```

Przykłady

```
GET / HTTP/1.1
Host: www.freebsd.org
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-
US; rv:1.7.7) Gecko/20050414 Firefox/1.0.3
Accept:text/xml,application/xml,application/xhtml+xml,...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
If-Modified-Since: Mon, 09 May 2021 21:01:30 GMT
If-None-Match: "26f731-8287-427fcfaa"
```

Przykłady

```
HTTP/1.1 200 OK
Date: Fri, 13 May 2020 05:51:12 GMT
Server: Apache/1.3.x LaHonda (Unix)
Last-Modified: Fri, 13 May 2020 05:25:02 GMT
ETag: "26f725-8286-42843a2e"
Accept-Ranges: bytes
Content-Length: 33414
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

HTTP - pola nagłówka ogólnego przeznaczenia (General)

- **`Date`** (Data) - Zapis daty i czasu powinien być zawsze zgodny z formatem podanym w RFC, inaczej ryzykujemy niepoprane działanie mechanizmów cache oraz cookie
- **`Transfer-Encoding`** - nie mylić z Content-Encoding, odnosi się do całego transferu i może przyjąć tylko wartość - "chunked"
- **`Cache-control, Via`** - zob. Serwery proxy

HTTP – Transfer Encoding

- Transfer-encoding **chunked** pozwala na “stopniowe” przesyłanie generowanych danych gdy z góry nie jest znana ich długość

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

A – długość danych (szesnastkowo, dziesiętnie 10)
0123456789 CR LF CR LF

14 – długość danych (szesnastkowo, dziesiętnie - 20)
01234567890123456789 CR LF CR LF

0
```

HTTP KeepAlive

- Zestawianie jednego połączenia dla każdego polecenia HTTP jest b. niewydajne:
  - Typowy przykład: pobierana jest strona WWW, a następnie szereg zawartych w niej plików graficznych - otwieranie sesji TCP za każdym razem jest b. kosztowne.

- Rozwiązanie: przesyłanie wielu poleceń HTTP jednym połączeniem TCP
  - klient ustawia opcję nagłówka HTTP "**Connection: Keep-alive**", jeśli serwer ją obsługuje - to samo połączenie może być wykorzystane wiele razy

- Opcja alternatywna: "**Connection: close**"

HTTP - pola nagłówka żądania

- Accept: przyjmowane typy danych:

```
accept: text/*, image/gif, image/jpg
```

- Accept-Charset:

```
Accept-charset: ISO-8859-1
```

- Accept-Encoding (dozwolone: gzip, compress, deflate (LZW)):

```
Accept-encoding: gzip
```

- Kodowanie znaków (charset, character encoding)
  - Zgodne z MIME
  - w szczególności:
    US-ASCII, ISO-8859-*, UNICODE-1-1, UNICODE-1-1-UTF-7, UTF-8

- Accept-Language (zgodnie z kodami ISO639):

```
Accept-Language: en,pl
```

HTTP - pola nagłówka żądania i odpowiedzi (wybrane)

request

`User-Agent` - nazwa i wersja przeglądarki, pozwala na dynamiczne dostosowanie zwracanej zawartości do przeglądarki

`Referer` - poprzedni URL, b. przydatne w statystykach, pozwala na określenie tzw. ścieżki klienta

`Host` - **wymagane**, pozwala na określenie serwera w przypadku gdy do jednego adresu IP przypisano wiele serwerów wirtualnych

```
GET /pub/WWW/ HTTP/1.1
Host: www.w3.org
```

response

`Server` - nazwa i wersja serwera (proxy dokłada "via:")

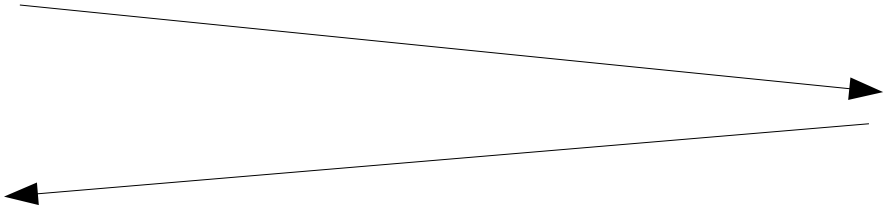
`Location` - przekierowania na inny URL, HTTP – redirect 30x

Poprzedni URL i pole Referer

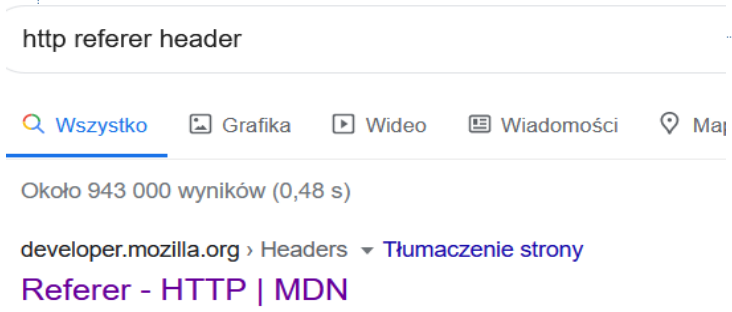
Dokument o adresie
http://site1.com/url1.html

http://site2.com/

```
GET /url2 HTTP 1.1
Referer: http:/site1.com/url1.html
```



Zapytanie do Google:
`http referer header`
otrzymujemy:
https://www.google.com/search?q=http+referer+header

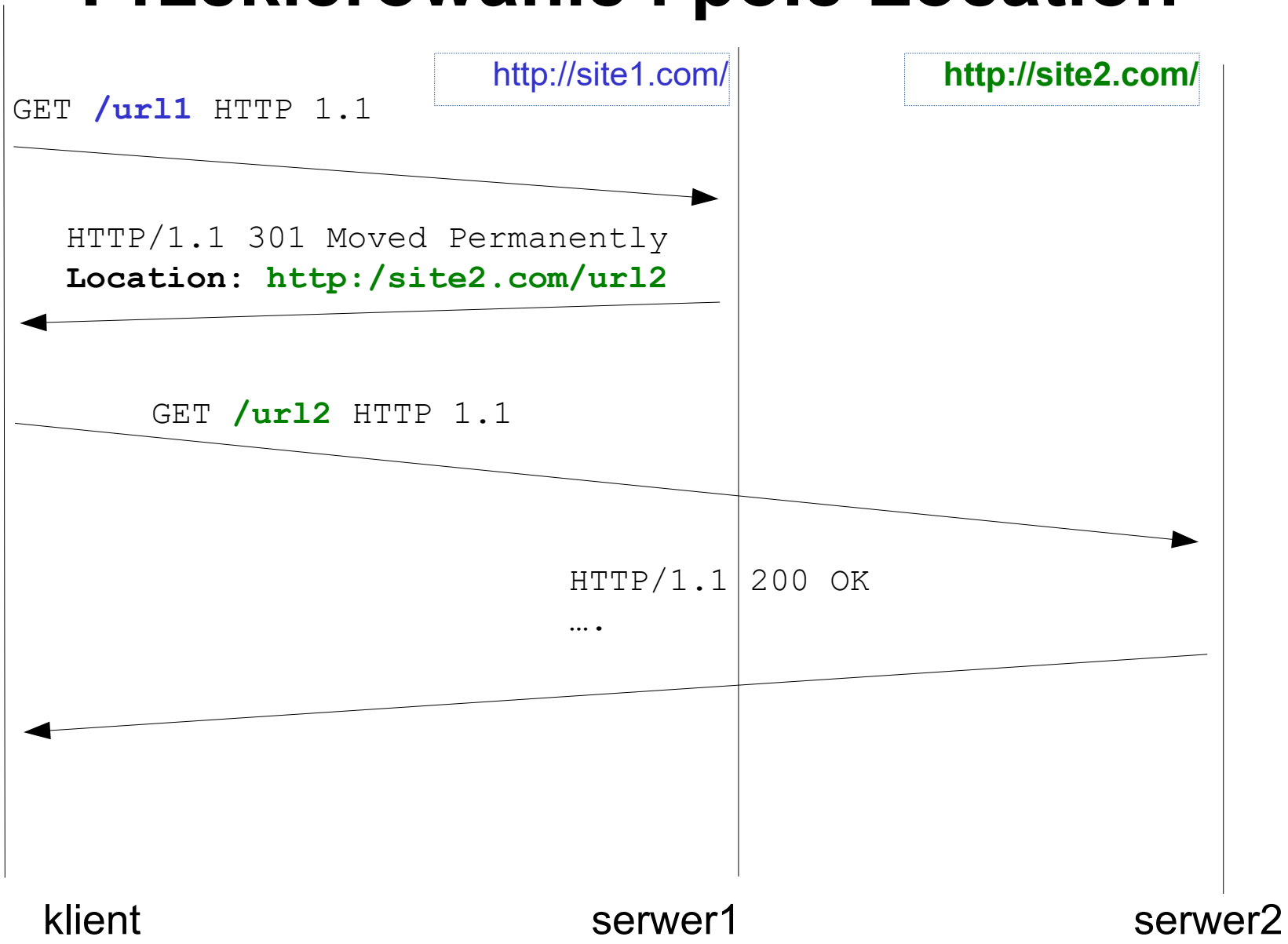


https://developer.mozilla.org/

```
GET /en-US/docs/Web/HTTP/Headers/Referer HTTP 1.1
Referer: https://www.google.com/
         search?q=http+referer+header
```

klient serwer

Przekierowanie i pole Location



HTTP Entity header (zawartość)

- Pola Entity (nagłówka zawartości):
  - **Entity-Header** zawiera opis przesyłanej treści (np. zgłaszanego formularza, zwracanego dokumentu)
  - **Content-Type** - informacja o typie treści, zgodna z MIME, typowo: "Content-Type: text/html"
  - **Content-Length** - długość Entity-Body w bajtach (dla HEAD jakby to było GET)
    - Musi być wyliczona z dokładnością do bajta - Uwaga - szczególnie ważne przy keep-alive
  - **Content-Encoding:** pozwala zawrzeć informację o kompresji, np: Content-Encoding: x-gzip
  - **Content-Disposition**: pozwala zadysponować co zrobić z obiektem (np. zapisać jako plik):
    Content-Disposition: Attachment; filename=example.html
  - Inne: **Content-Language, Content-MD5, Expires, Last-Modfied**

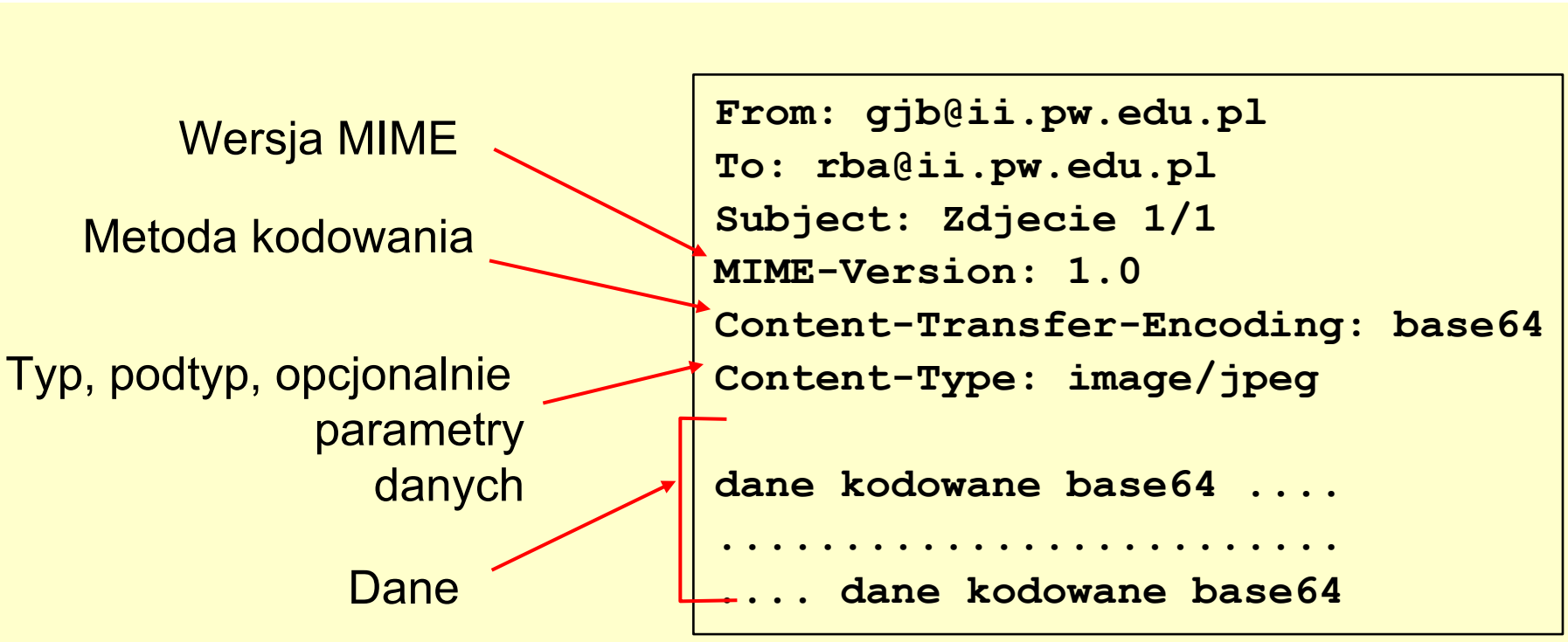
MIME
(Multipurpose Internet Mail Extensions)

MIME - podstawy

RFC2045

RFC2046

- **MIME** - aktualny standard RFC 2045, 2046 (pierwsze wersje RFC1341 - 1992)
- Dodatkowe pola nagłówka definiują format danych
- **Uwaga** - MIME pomyślany dla e-mail, jednak aktualne zastosowanie wykracza poza SMTP! (np. HTTP, XML, ...)



Podstawowe typy MIME

Content-type: type/subtype; parameters

Text

- **`text/plain, text/html`**
  `text/plain; charset=us-ascii`

Image

- **`image/jpeg, image/png`**

Audio

- **`audio/basic`**
  jeden kanał, 8bit PCM 8000 Hz

Video

- **`video/mpeg, video/quicktime`**

Application

- **`application/msword, application/octet-stream`**

Złożone typy MIME

Content-Type: multipart/mixed; boundary=*bndr-string*

- Wiadomość podzielona na wiele części oddzielonych przez unikalny (i arbitralny) ciąg znaków "boundary string"
- Generowanie boundary string – heurystyczne (?)
- Każda część może mieć własny content-type, np:
  - pierwsza część może być typu text/plain
  - druga typu image/gif kodowana base64

MIME Multipart/mixed - przykład

```
From: <donald.duck@disney.com>
To: <mickey.mouse@disney.com>
Subject: blah
MIME-Version: 1.0
Content-Type: multipart/mixed;  boundary= "Boundary-
   00=_9W2T/VtQiQcNR1P"

--Boundary-00=_9W2T/VtQiQcNR1P
Content-Type: text/plain; charset=US-ASCII

tekst tekst tekst tekst tekst tekst tekst tekst

--Boundary-00=_9W2T/VtQiQcNR1P
Content-Type: application/octet-stream
...
```

MIME Multipart/mixed - przykład

```
From: <donald.duck@disney.com>
To: <mickey.mouse@disney.com>
Subject: blah
MIME-Version: 1.0
Content-Type: multipart/mixed;  boundary= "Boundary-
   00=_9W2T/VtQiQcNR1P"
...
--Boundary-00=_9W2T/VtQiQcNR1P

Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename= "nazwa.bin"

base64base64base64base64base64...
base64base64base64base64base64...
--Boundary-00=_9W2T/VtQiQcNR1P--
```

Boundry poprzedzone jest --

Ostatnie boundry zakończone jest --

Inne złożone typy MIME

- **`multipart/alternative`** - ta sama treść przesłana w kilku wariantach, np. w formacie text i HTML, klient wybiera właściwą postać do prezentacji danych.

- **`multipart/parallel`** - format identyczny jak w mixed, służy do równoległej prezentacji danych w kilku formatach.

- **`multipart/signed, multipart/encrypted –`** wiadomość podpisana / zaszyfrowana (S/MIME – RFC 1847)

Kodowanie Base64

- Proste kodowanie pozwalające na przesłanie dowolnych danych binarnych w postaci "drukowalnych" znaków ASCII (7bit)
- Wielkość danych zwiększa się o 4/3, np. plik o rozmiarze 3KB ma po zakodowaniu 4KB
- Bloki 3 bajtowe zamieniane są na 4 liczby 6-o bitowe
- Każda liczba 6-o bitowa zamieniana jest na znak drukowalny z przedziału: A-Za-z0-9+/
- Jeśli wielkość danych kodowanych w bajtach nie jest podzielna przez 3 - uzupełnienie zerami/znakami = na końcu
- Wynikowy tekst jest dzielony na linie o długości 76 znaków
- Tak otrzymany tekst jest do zaakceptowania przez każdy MUA/MTA zgodny z podstawowym SMTP

Więcej o MIME

- **RFC:**
  - **RFC 2045** - nagłówki definiowane w standardzie MIME
  - **RFC 2046** - struktura danych MIME i podstawowe typy danych w MIME
  - **RFC 2047** - rozszerzenia MIME w treści nagłówków wiadomości
  - **RFC 2048**, - procedury rejestracji nowych typów MIME w IANA
  - **RFC 2049** - reguły zgodności aplikacji z MIME oraz przykłady

MIME - kodowanie

- Zdefiniowane są następujące typy "Content-Transfer-Encoding":
  - "7bit"
  - "8bit"
  - "binary"
  - "quoted-printable"
  - "base64"
  - ietf-token oraz x-token

nie dokonano przekodowania

Wiadomość jest kodowana w specjalny sposób

- Typ "quoted-printable" - do kodowania wiadomości składających się głównie, ale nie wyłącznie, ze standardowych znaków ASCII
- **Uwaga:** RFC 2045 określa dozwolone typy kodowania dla złożonych typów danych (multipart i message) jako wyłącznie: 7/8 bit oraz binary

MIME - "quoted-printable"

- Kodowanie mające na celu zachowanie danych (zwykle tekstu) bez modyfikacji nawet przez serwery niezgodne ze standardami
- każdy oktet może być reprezentowany jako: **=hh**, gdzie **h** - cyfra szesnastkowa: 0123456789ABCDEF
- oktety o kodach: 33-60 (większość znaków przestankowych) oraz: 62-126 (litery, cyfry, [,] , |, ~) mogą być reprezentowane bezpośrednio
- białe znaki reprezentowane są bezpośrednio, **chyba**, że są na **końcu linii**
- znak **=** na końcu linii oznacza "soft line break" (można łamać długie linie)
- Przykład:

```
Dzia=B3aj=B1c w imieniu i na rzecz=
          sp=F3=B3ki "Zielone buraczki" Sp. =
```

RFC 2047 - "Message Header Extensions"

- Do tej pory omówione rozwiązania nie pozwalają na zawarcie znaków **nie ASCII** w nagłówku - co może być przydatne np. w w nagłówkach, które mogą zawierać tzw. znaki narodowe
- RFC 2047 rozwiązuje ten problem wprowadzając tzw. "encoded word"
- **Uwaga** – quoted printable może być też stosowane w entity-body

```
encoded-word = "=?" charset "?" encoding "?" encoded-text "?="
```

- Charset - określa kodowanie (np. ISO-8859-2)
- encoding - **Q** - quoted printable, **B** - base64
- encoded-text - kodowany tekst

Przykład:

```
From: =?ISO-8859-1?Q?Olle_J=E4rnefors?= <ojarnef@admin.kth.se>
Subject: =?ISO-8859-1?B?SWYgeW91IGNhbiByZWFkIHRoaXMgeW8=?=
    =?ISO-8859-2?B?dSB1bmRlcnN0YW5kIHRoZSBleGFtcGxlLg==?=
```

RFC 2049 - Zgodność z MIME

- **RFC 2049**: *"[...] There exist many widely-deployed non-conformant MTAs in the Internet. These MTAs, speaking the SMTP protocol, alter messages on the fly to take advantage of the internal data structure of the hosts they are implemented on, or are just plain broken. [...]"*
- RFC 2049 - określa minimalny poziom zgodności, w szczególnosci sytuacje gdy program natrafi na nieznany typ danych (zalecenie - traktować jak *multipart/mixed*), lub nieobsługiwany zestaw znaków (zalecenie - traktować jak application/octed-stream)
- RFC 2049 precyzuje standard, określa kroki jakie musi wykonać aplikacja generując wiadomość MIME

HTTP - Range

HTTP/1.1 – RFC2616
HTTP/1.1 – RFC7233

- Nagłówek pozwala na transfer fragmentu obiektu
- Przydatne przy dużych obiektach (multimedia)
- Zakresy mogą być indywidualnie cachowane
- Nagłówki: Range, If-Range, Content-Range
- Przykłady:

```
C:
Range: bytes 100-1200

S:
HTTP/1.1 206 Partial Content
Content-Range: bytes 100-1200/1201
Content-Length: 1101

HTTP/1.1 416 Range Not Satisfiable
Content-Range: bytes */1201
```

Autoryzacja w HTTP

Autoryzacja:
RFC 7235
+ szereg uzupełnień
oraz: RFC 2817

- Autoryzacja dotyczy "**realms**" - arbitralnych grup zasobów serwera, typowo będzie to katalog z podkatalogami (ale nie koniecznie)
- Serwer żąda autoryzacji: zwraca kod **401** oraz nagłówek: **`WWW-Authenticate:`** ***`challenge`***

```
challenge: auth-scheme "realm=name" *( "," auth-param )
```

Klient powinien poprawnie odpowiedzieć na *challenge:*

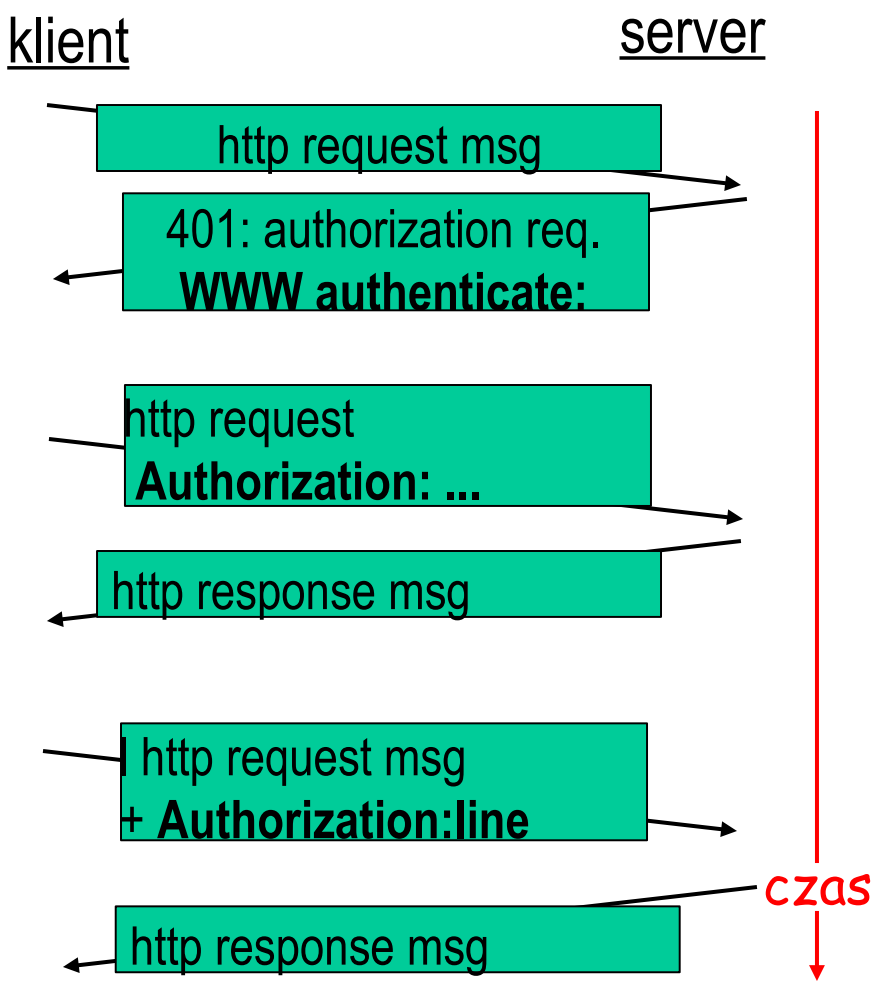
```
S: WWW-Authenticate: Basic realm="WallyWorld"
K: Authorization: Basic QWxhZGRpbjpvcGVHNlc2FtZQ==
```

Tryb autoryzacji "Basic" - klient zwraca Mime(userid:password)

Autoryzacja w HTTP

Dodatkowe uwagi:

- HTTP jest bezstanowy - za każdym razem gdy autoryzacja jest wymagana klient musi się uwierzytelnić
- po pierwszej autoryzacji klienta (wpisanej "ręcznie") przeglądarka będzie prowadzić autoryzację automatycznie
- istnieje też autoryzacja do proxy: `Proxy-Authorization`



Autoryzacja w HTTP

Dodatkowe uwagi:

- Podstawowa specyfikacja HTTP nie określa schematów uwierzytelnienia
- Znajdziemy je tu:

  http://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml
- Istnieje też autoryzacja do proksy, która funkcjonuje podobnie do opisanej aut. podstawowej: `Proxy-Authorization`

HTTP Authentication Schemes

Registration Procedure(s)
IETF Review
Reference
[RFC7235]
Available Formats
CSV

Authentication Scheme Name	Reference	Notes
Basic	[RFC7617]	
Bearer	[RFC6750]	
Digest	[RFC7616]	
HOBA	[RFC7486, Section 3]	The HOBA sch fields are used
Mutual	[RFC8120]	
Negotiate	[RFC4559, Section 3]	This authentica syntax).
OAuth	[RFC5849, Section 3.5.1]	
SCRAM-SHA-1	[RFC7804]	
SCRAM-SHA-256	[RFC7804]	
vapid	[RFC 8292, Section 3]	

Autoryzacja HTTP Ch-Rp

- **MD5 –** message digest, zasada taka sama jak “basic”, ale z pary (userid:password) zamiast kodowania base 64 tworzony jest skrót MD5
- MD5 jest wrażliwy na tzw “replay attack”
- Atakowi “Replay” zapobiega typ autoryzacji “**Digest**”:
  - Serwer wraz z 401 przysyła unikalną wartość: “nonce”
  - Klient oblicza:

```
A1 = MD5 (username +":" + realm + ":" + password)
A2 = MD5 (method + ":" + uri)
requestdigest = MD5 (A1 + ":" + nonce + ":" + A2)
```

Serwer może też obliczyć spodziewaną odpowiedź klienta i sprawdzić czy jest poprawna

Przykład autoryzacji MD5

```
C: GET /index.html HTTP/1.1
C: Host: localhost

S: HTTP/1.0 401 Unauthorized
S: WWW-Authenticate: Digest realm="testrealm@host.com",
   qop="auth,auth-int",nonce="dcd98b7102dd2f0bfb0c093",
   ...
```

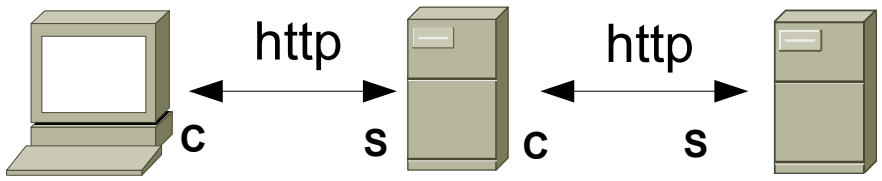
HA1=MD5(username:realm:password)
HA2=MD5(method:digestURI:MD5(entityBody))
response=MD5(HA1:nonce:nonceCount:clientNonce:qop:HA2))

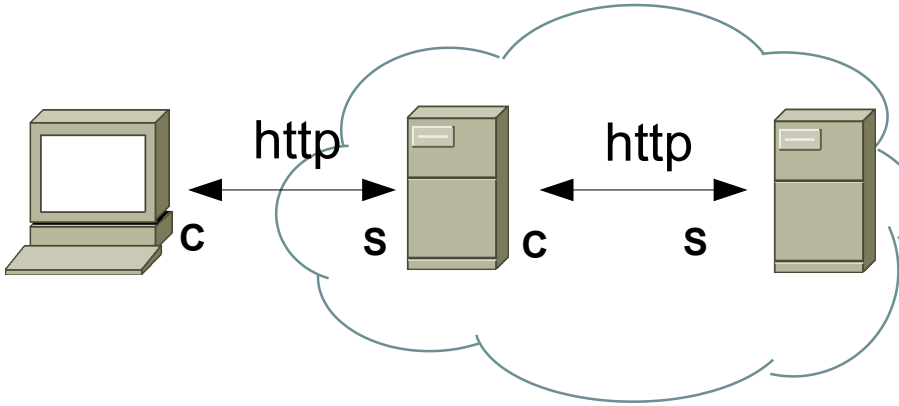
```
C: GET /index.html HTTP/1.1
C: Authorization: Digest username="Mufasa",
   realm="testrealm@host.com",
   nonce="dcd98b7102dd2f0bfb0c093",
   uri="/index.html",qop=auth, nc=00000001,
   cnonce="0a4f113b",
   response="6629fae49393a05397450978507c4ef1"
```

Proxy i cache HTTP

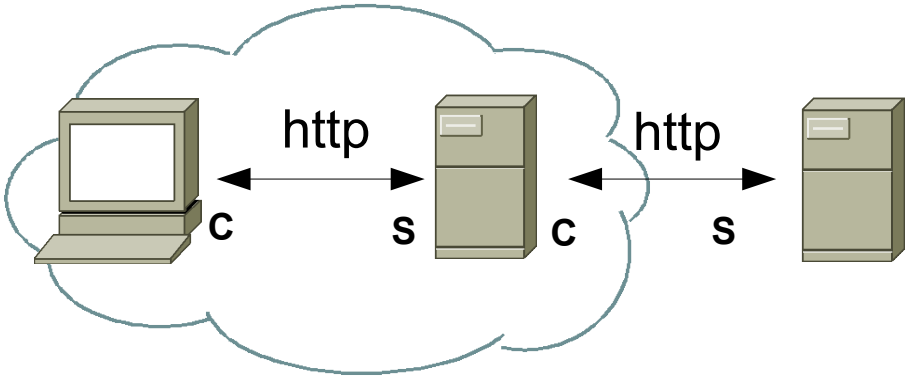
- Proxy - serwer pośredniczący
  - dla klienta jest serwerem
  - dla serwera jest klientem
  - cele:
    - separacja klientów od ”świata zewnętrznego" (bezpieczeństwo)
    - autoryzacja “wewnętrzna”
    - filtrowanie treści przesyłanych z internetu
    - odporność na (złośliwe) anomalie protokołu
    - zwiększenie szybkość pracy
- Cache - specyficzny serwer proxy - jego celem jest przyśpieszenie działania poprzez przeniesienie treści "bliżej" odbiorcy.
  - **Uwaga** – lokalny cache przeglądarki zachowuje się podobnie do serwera proxy!

Proxy i cache HTTP





Odwrotne (reverse) proxy



Proxy w organizacji

Proxy i cache HTTP

- Klient może zwracać się do serwera pośredniego Proxy i pobierać dane z niego
- Zalety i korzyści:
  - jeśli dane są już w cache:
    - skrócenie czasu odpowiedzi
    - oszczędność pasma, odciążenie serwera
  - bezpieczeństwo - cache może dodatkowo filtrować dane
  - Inne: konwersja prot. np.: http-https
- Poziomy cache:
  - *przeglądarka*
  - serwer w organizacji
  - serwer w sieci szkieletowej

Proxy/cache HTTP

- Specyfikacja rozróżnia:
  - "caching proxy" i "non caching proxy"
  - "transparent proxy" i "non transparent proxy"
- Transparent – tylko modyfikacje nagłówka związane z obsługą proxy
- Non-Transparent – inna ingerencja w treść
- Niektóre pola nagłówka przetwarzane są w trybie "hop-by-hop" (a nie end-to-end), m.in.:
  - Connection, Keep-Alive, Proxy-Authenticate, Proxy-Authorization, TE, Trailers, Transfer-Encoding, Upgrade
  - Dostępnych jest szereg mechanizmów pozwalających na określenie przez klienta i serwera "aktualności" dokumentu

Co nie podlega cachowaniu?

- Zazwyczaj rezultaty wygenerowane dynamicznie
- Dane szyfrowane SSL/TLS - (dlaczego?)
- Dane personalizowane - cookies
- A także …. dokumenty nieaktualne - jak określić aktualność?
  - czas (jakie mogą się pojawić problemy?)
  - unikalny identyfikator

HTTP i cache

- **Expires**: informacja do kiedy Entity-Body jest ważne
- **Last-Modified** - zwraca serwer, data ostatniej modyfikacji obiektu odpowiadającego Entity-Body
  - `Last-Modified = "Last-Modified" ":" HTTP-date`
- **If-Modified-Since** - klienckie, używane razem z GET, jeśli nie modyfikowane to zwrócone zostanie 304 (not modified), inaczej zwrócone 200 oraz aktualne entity-body
  - `If-Modified-Since = "If-Modified-Since" ":" HTTP-date`
  - Jeśli nie modyfikowane: `304 Not Modified`
  - `If-Modified-Since: Mon, 8 Dec 2014 19:43:31 GMT`
- **If-Unmodified-Since –** klienckie
- **If-Range** – jeśli Last-/If- spełnione to odesłać podany zakres

HTTP Cache-control - dyrektywy dla serwera proxy/cache

Cache-control: `"cache-control: no-cache", ...`

- **Serwerowe:**
  - **`public, private`** - podlega cachowaniu, podlega cachowaniu ale tylko w "prywatnym" cache użytkownika
  - **`no-cache`** – proksy: zawsze wymaga rewalidacji (klienckie: wymusza akt. wersję)
  - **`no-store`** - bezwzględny zakaz zapisu na dysku (klienckie: nie używaj cache)
  - **`only-if-cached`** - kopia powinna być aktualizowana możliwie rzadko (b. wolne łącze proxy - serwer źródłowy)
- **… i Klienckie:**
  - **`No-cache:`** wymusza aktualizację wersji (proksy ma pobrać dane z sewera)
  - **`max-age=seconds`** - nie wysyłać informacji starszych niż tu określono - odnowić jeśli zadany czas przekroczony (mocniejsze od Expires)
  - **`min-fresh=seconds`** - nie wysyłać informacji, które będą aktualne krócej niż określono (K)
  - **`max-stale=seconds`** - można przekroczyć wygaśnięcie ważności (patrz wyżej) o zadaną liczbę sekund bez konieczności odnawiania treści dokumentu (K)

HTTP i cache - nagłówek

HTTP/1.1 200 OK

Date: Sat, 1 May 2021 14:10:01 GMT

Server: Apache/1.3.3 (Unix)

Cache-Control: max-age=3600, must-revalidate

Expires: Sat, 1 May 2021 14:19:41 GMT

Last-Modified: Fri, 30 Apr 2021 02:28:12 GMT

ETag: "3e86-410-3596fbbc"

Content-Length: 1040

Content-Type: text/html

Cache może wysyłać odp. do klientów, ale musi ponownie skontaktować się z serwerem, gdy ważność upłynie.
proxy-revalidate - nie odnosi się do lokalnego cache

HTTP i cache

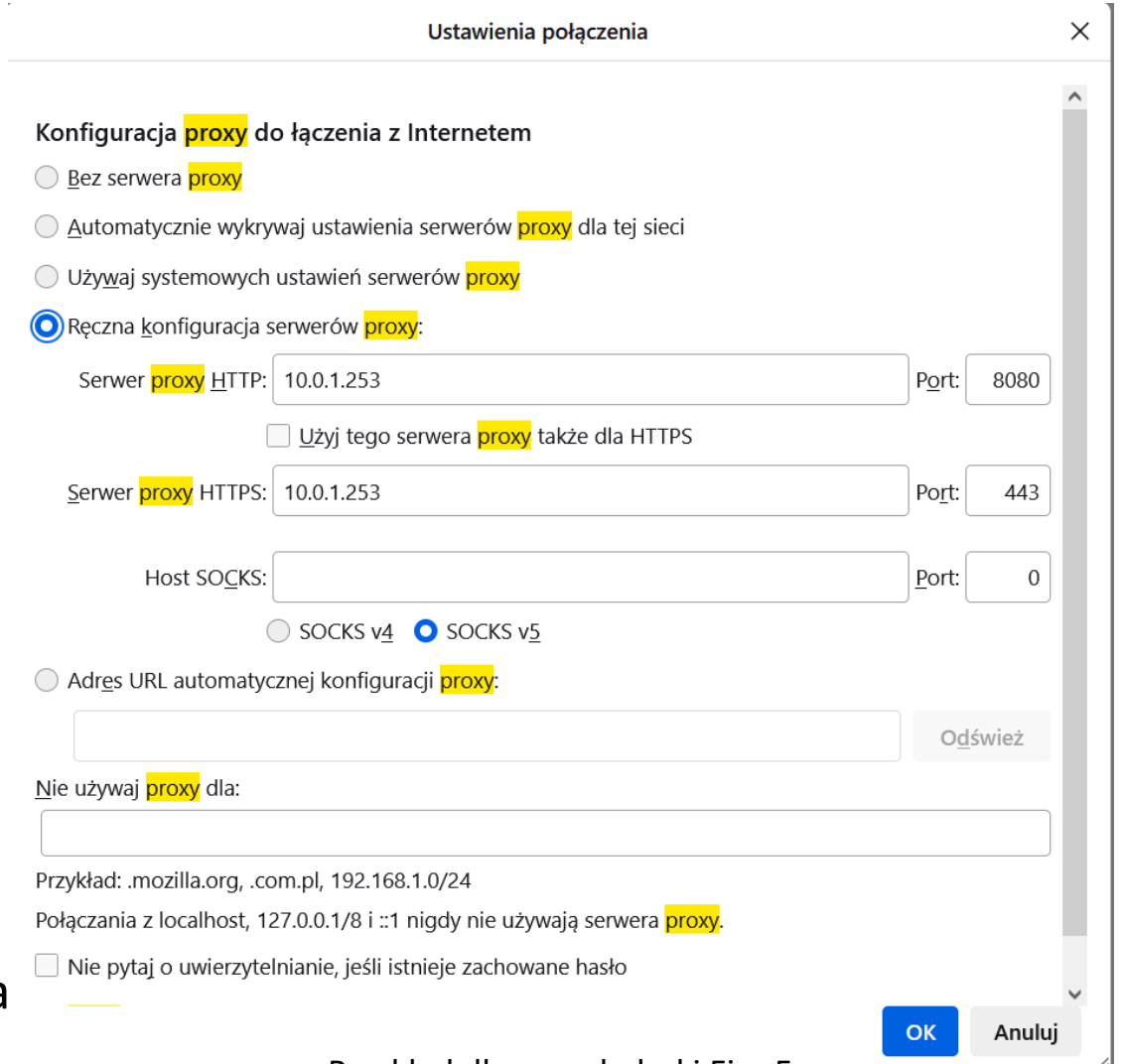
- Entity Tag - używany do porównania entity przypisanych do tego samego URL
- Entity - **`ETag: W/"xyzzy"`**
  - przyznawany przez serwer
  - odsyłany lecz nieinterpretowany przez klienta
  - silne lub słabe (weak: */W*, strong) - jakakolwiek zmiana, zmiana "istotna" - np. przydatne dla liczników odwołań
- Nagłówki klienckie: If-Match, If-None-Match razem z: If-Modified-Since, If-Unmodified-Since, If-Range pozwalają na pobranie wersji aktualnej:
  - **`If-Match: "xyzzy"`**
- Tag typowo obliczany jako szybki hash

Efektywne wykorzystanie mechanizmów cache

- Konsekwentne wykorzystanie URL-i (nie zmieniać, nie duplikować).
- Wykorzystanie biblioteki grafiki, skryptów, itd.
- Stosować max-age dla tych danych, które rzadko się zmieniają (np. podstawowe elementy graficzne).
- W przypadku zmiany treści obiektu o dużym max-age:
  - zmienić nazwę, a nie zawartość (tak aby max-age nie był fałszywy),
  - przy przenoszeniu plików użyć "move" a nie "copy".
- Ograniczyć wykorzystanie:
  - Cookies,
  - TLS,
  - POST w skryptach (nie cachowane).

Jak klient korzysta z proxy

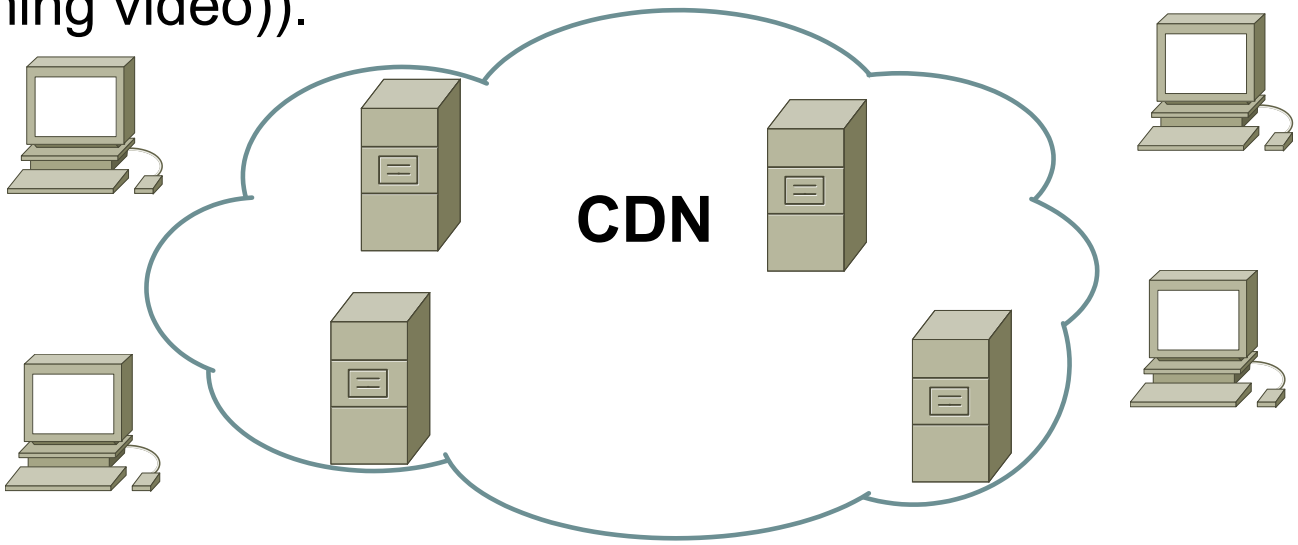
- Adres serwera proxy:
  - Może być skonfigurowany jawnie przez użytkownika
  - Może być wymuszony administracyjnie
  - Może być wykryty automatycznie (prot. WPAD bazujący na DHCP)
  - Dla niektórych (popularnych) protokołów (np. HTTP) parametry serwera proxy mogą być pobrane poprzez odpowiedni plik konfiguracyjny
  - Protokół SOCKS pozwala na przesyłanie przez proxy arbitralnych protokołów



Przykład dla przeglądarki Fire Fox

CDN – Content Delivery Network

- Rozproszony terytorialnie (PoPs) system serwerów dostarczających dane do klientów (typowo HTTP).
- Cele: niezawodność, szybkość działania (responsywność), bezpieczeństwo (odporność na DdoS).
- Z punktu widzenia wlaściciela serwisu jest to usługa.
- Technologie: L3-L7 (load-ballancing,content switch, multilayer switch, protokoły: ICAP, OPES; P2P; prywatne CDN; operatorskie CDN (streaming video)).



HTTP cookie

HTTP cookie:
RFC6265
RFC2965 (stare)

- Prosty mechanizm utrzymania stanu sesji.
- Serwer przekazuje klientowi daną - "Cookie" nie interpretowaną przez klienta – nagłówek:
  “Set-cookie”.
- Klient zwraca tę samą daną, pod warunkiem: zgodności serwera, ścieżki oraz aktualności cookie; nagłowek: “Cookie”:

```
Set-Cookie: $name=$value[; expires=$date][; path=$path]
   [; domain=$domain][; Secure][; HttpOnly]
```

```
S: Set-Cookie: SID=31d4d96e442
C: Cookie: SID=31d4d96e442
S: Set-Cookie: SID=31d4d96e442; Path=/; Secure; HttpOnly
C: Cookie: SID=31d4d96e442; lang=en-US
```

HTTP cookie

Klient może zwrócić kilka cookie:

```
S:  Set-Cookie: Part_Number=Rocket_Launcher_0001;
    Path=/acme
C: Cookie: Customer=WILE_E_COYOTE; Path=/acme;
    Part_Number=Rocket_Launcher_0001; Path=/acme

S: Set-Cookie: Part_Number=Rocket_Launcher_0002;
    Path=/acme; Domain=www.acme.com
C: Cookie:
    Path=/acme;Part_Number=Rocket_Launcher_0002"
    Path=/acme; Domain=www.acme.com
```

HTTP cookie – ważność i bezpieczeństwo

- Cookie ma termin ważności.
- Po upływie terminu cookie jest usuwane.
- Jeśli nie podano terminu ważności, cookie powinno zostać usunięte przy zamykaniu programu przeglądarki.
- Cookie może też mieć „max-age”
- Ustawienie max-age=0 powoduje skasowanie cookie
- Przykład:
  - **S:** `Set-Cookie: ID=732343431173242;` **`expires=Fri, 31-Dec-2010 23:59:59 GMT;`** `path=/; domain=.example.com;`
- Atrybut Secure – cookie będzie ustawione tylko w prot. https
- Atrybut HttpOnly – cookie nie będzie dostępne poza http (np. z poziomu skryptu)

HTTP cookie

- Niezgodność FQDN lub path z URI powoduje odrzucenie cookie przez klienta:
  - FQDN są zgodne lub są identycznymi adresami IP - OK
  - zgodność składnika domenowego, tzn: host.xyz.com.pl i xyz.com.pl zgodne, host.abc.com.pl i com.pl NIE
  - domain=.example.com; path=”/” ustawi cookie dla każdego serwera i scieżki z domeny j.w.
- Klient powinien obsługiwać (RFC 2109) :
  - do 300 cookies
  - do 4096 B na cookie
  - do 20 cookie na hosta/domenę (FF: 50, IE: 20/50)

HTTP cookie

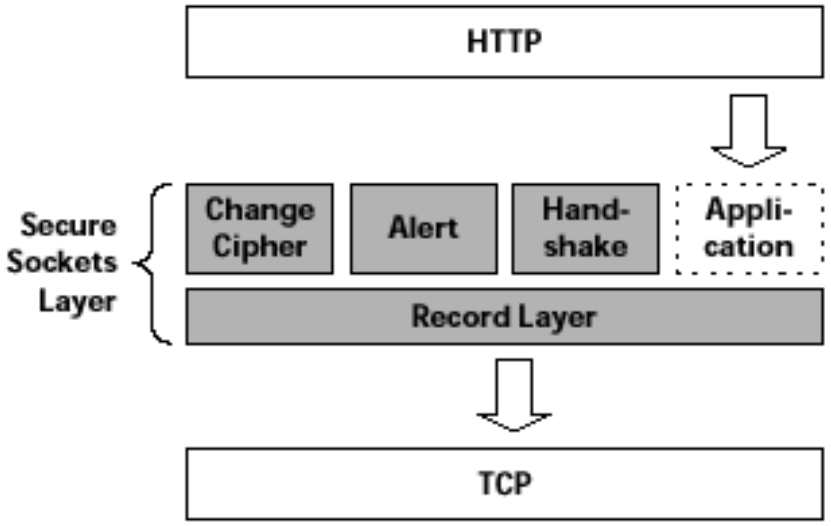
- Jeden URL może generować wiele cookie.
- Cookie jest związane **zawsze** z danym adresem/domeną (ważne dla bezpieczeństwa serwera i klienta).
- Cookie jest daną - nie programem, i o ile nie ma błędu związanego z jego obsługą w przeglądarce, nie stanowi zagrożenia dla bezpieczeństwa.
- Z cookie są związane zagrożenia dla prywatności.
- Znane klasy problemów / ataków:
  - Cookie hijaking - podsłuch, podmiana
  - Cross-site cooking – podmiana między site-ami
  - Cookie poisoning - podmiana otrzymanego cookie

Cookies - zastosowania

- Zarządzanie stanem – budowa sesji (w obrębie aplikacji Web):
  - Informacja o użytkowniku, np. preferowany język interfejsu,
  - Stan sesji, np. koszyk zakupów w e-sklepie.
- Śledzenie zachowania użytkowników:
  - W obrębie jednej domeny / aplikacji.
    - Dużo elastyczniejsze niż “refer”
  - W obrębie wielu domen (cross-site). Jak? - Poprzez zewnętrznego “brokera” dla cookie, np. serwer bannerowy.
  - Cross -site cookie mogą zostać zablokowane.

TLS (SSL) - bezpieczny prot. "transportowy"

- SSL v. 1/2 wprowadzony w przeglądarce Netscape na początku lat 90-tych (zawierał dość poważne błędy koncepcyjne)
- SSL v. 3 – opracowany od podstaw, 1996
- TLS (Transport Layer Security) 1.0 == SSL 3.0 z minimalnymi modyfikacjami (TLS przedstawia się jako SSL 3.1) 1999
- TLS - RFC 2246; http://www.openssl.org/ - "bazowa" darmowa implementacja
- TLS 1.2 – 2008; TLS 1.3 - 03/2018 RFC8466 – obecnie wymagany
- Sesja (session), połączenie (connection) -"wewnątrz" sesji
- Dwa "pod-protokoły":
  - "handshake" - inicjalizacja
  - "record" - dane



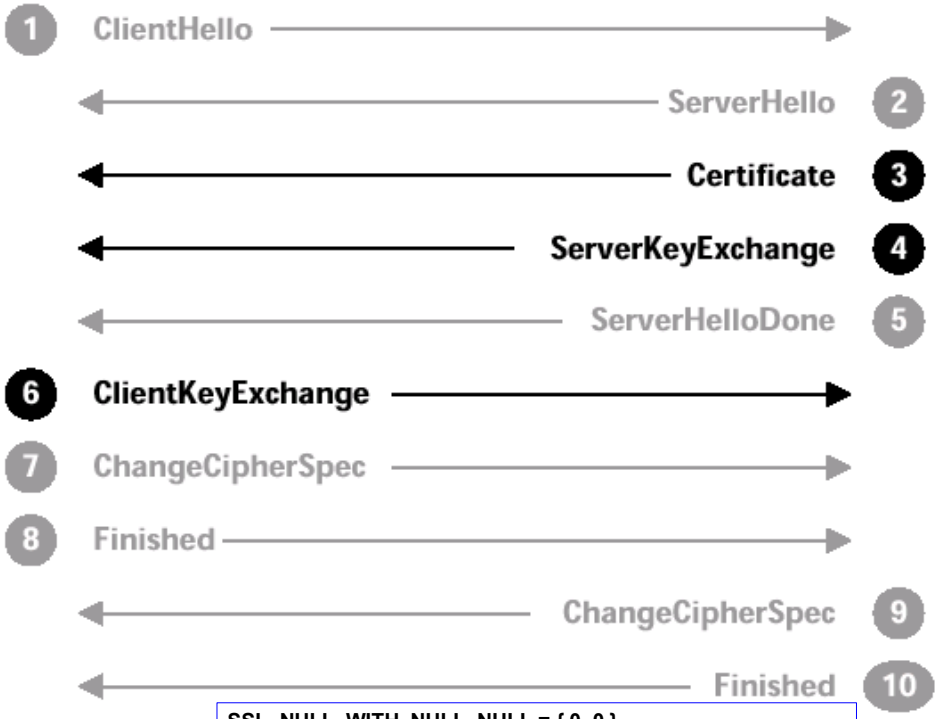
SSL / TLS

- Faza negocjacji połączenia SSL:
  - ustalenie metod szyfrowania i podpisu,
    - wybór algorytmu symetrycznego,
    - wybór metody wymiany kluczy algorytmu asymetrycznego,
    - wybór alg. skrótu (integralność).
  - Ustanowienie głównego klucza sesji (master secret).
  - Uwierzytelnienie serwera i opcjonalnie klienta:
    - uwierzytelnienie bazuje na certyfikatach X509v3 (klucz prywatny-klucz publiczny),
    - zaufanie do cert. wynika z jego podpisania przez nadrzędne CA (Certificate Authority) - “certificate chain”,
    - Certyfikat CA musi być “wbudowany” w serwer / klienta.

TLS 1.3

- TLS 1.2 – RFC 5246
- TLS 1.3 – RFC 8446
- Różnice między 1.3 i 1.2 są dość znaczne:
  - znacząca poprawa wydajności – zmniejszenie liczby wymian komunikatów (0-RTT) – z 7 do 5 (4-2) komunikatów c ↔ s
  - dodano nowe metody krypto
  - usunięcie przestarzałych metod krypto
  - eliminacja opcji zmniejszających bezpieczeństwo
  - pewne problemy z funkcjonowaniem proxy TLS

TLS – rozpoczęcie sesji



1: zestaw alg. krypto do wyboru
2: wybór alg.
3: certyfikat serwera do weryf.
4: klucz publiczny
5: koniec fazy negocjacji

6: zaszyfrowany klucz symetryczny
7: włączenie enkrypcji
8: koniec fazy negocjacji
9: włączenie enkrypcji
10: koniec negocjacji

przykładowe zestawy alg.

```
SSL_NULL_WITH_NULL_NULL = { 0, 0 }

SSL_RSA_WITH_NULL_MD5 = { 0, 1 }
SSL_RSA_WITH_NULL_SHA = { 0, 2 }
SSL_RSA_EXPORT_WITH_RC4_40_MD5 = { 0, 3 }
SSL_RSA_WITH_RC4_128_MD5 = { 0, 4 }
SSL_RSA_WITH_RC4_128_SHA = { 0, 5 }
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 = { 0, 6 }
SSL_RSA_WITH_IDEA_CBC_SHA = { 0, 7 }
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA = { 0, 8 }
SSL_RSA_WITH_DES_CBC_SHA = { 0, 9 }
SSL_RSA_WITH_3DES_EDE_CBC_SHA = { 0, 10 }
```

TLS – transport danych

dane aplikacji

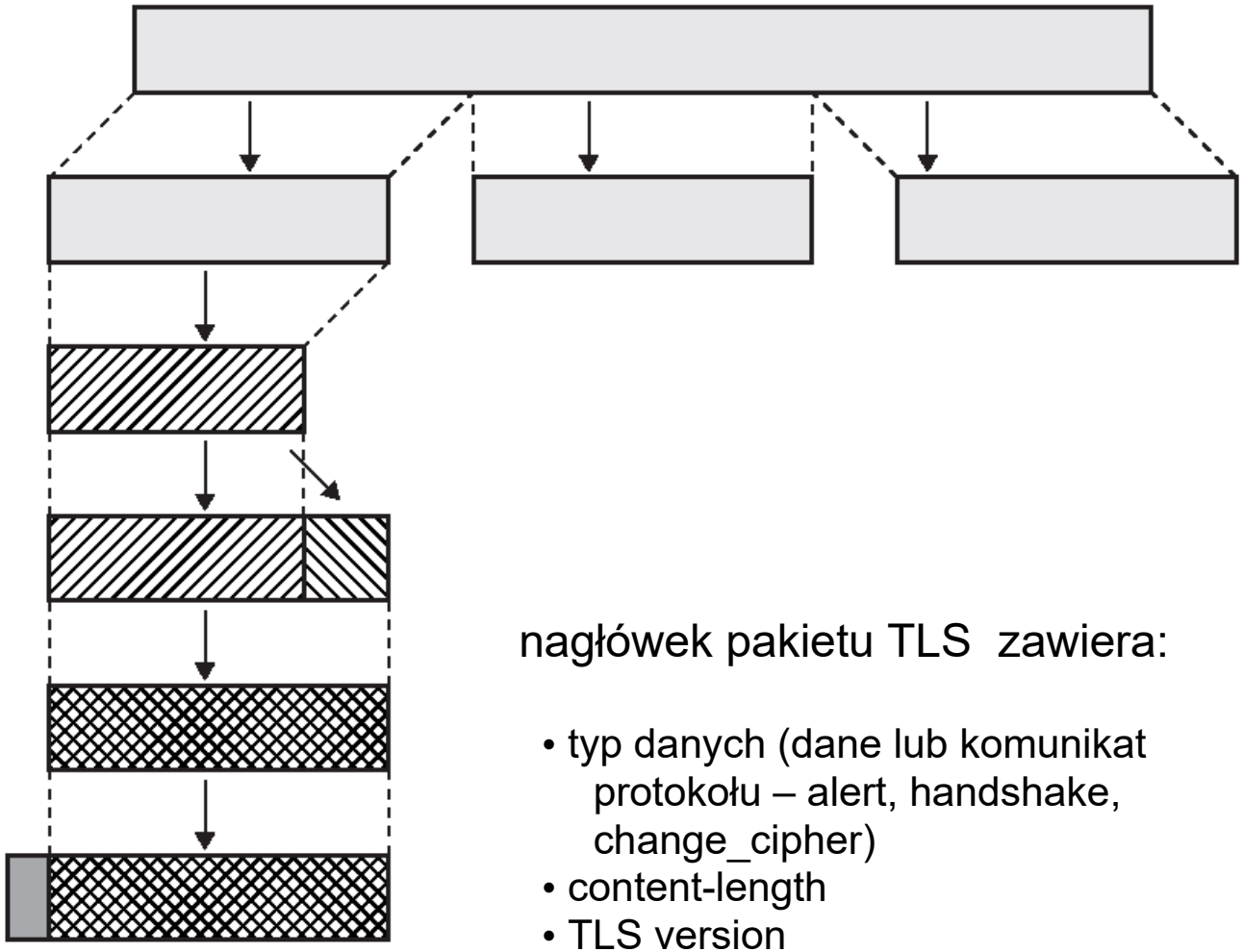
podział

kompresja

podpis (MAC)

szyfrowanie

nagłówek TLS



nagłówek pakietu TLS zawiera:

- typ danych (dane lub komunikat protokołu – alert, handshake, change_cipher)
- content-length
- TLS version

Certyfikat X509v3

Podgląd certyfikatu w przeglądarce

Certificate

online.mbank.pl	DigiCert SHA2 Extended Validation Server CA	DigiCert High Assurance EV Root CA

Nagłówek pokazuje nam łańcuch certyfikacji (“certificate chain” – od root CA, przez wystawcę, do właściwego certyfikatu)

Subject Name

Business Category	Private Organization
Inc. Country	PL
Serial Number	0000025237
Country	PL
State/Province	mazowieckie
Locality	Warszawa
Organization	mBank S.A.
Common Name	online.mbank.pl

Common Name - CN (albo dn) – na kogo wystawiony jest certyfikat – tzw. “Subject” lub “Persona” - tu adres www

Issuer Name

Country	US
Organization	DigiCert Inc
Organizational Unit	www.digicert.com
Common Name	DigiCert SHA2 Extended Validation Server CA

Certyfikat X509v3 c.d.

Validity

Not Before	Fri, 16 Jul 2021 00:00:00 GMT
Not After	Tue, 16 Aug 2022 23:59:59 GMT

Public Key Info

Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	E7:BD:1F:04:5C:79:DE:70:85:87:3C:AE:B5:C2:30:7E:03:19:5C:3C:FD:A3:D9:88:FA:1...

Basic Constraints

Certificate Authority	No

Key Usages

Purposes	Digital Signature, Key Encipherment

Extended Key Usages

Purposes	Server Authentication, Client Authentication

CRL Endpoints

Distribution Point	http://crl3.digicert.com/sha2-ev-server-g3.crl
Distribution Point	http://crl4.digicert.com/sha2-ev-server-g3.crl

Certyfikat zawsze ważny jest w określonym przedziale czasowym

Informacja o kluczu publicznym, klucz prywatny związany z cert. nigdy nie jest przekazywany i nie jest zapisany w certyfikacie.

Dozwolone zestosowania certyfikatu

CRL – Certificate Revocation List – adresy serwerów przechowujacych listy unieważnionych certyfikatów

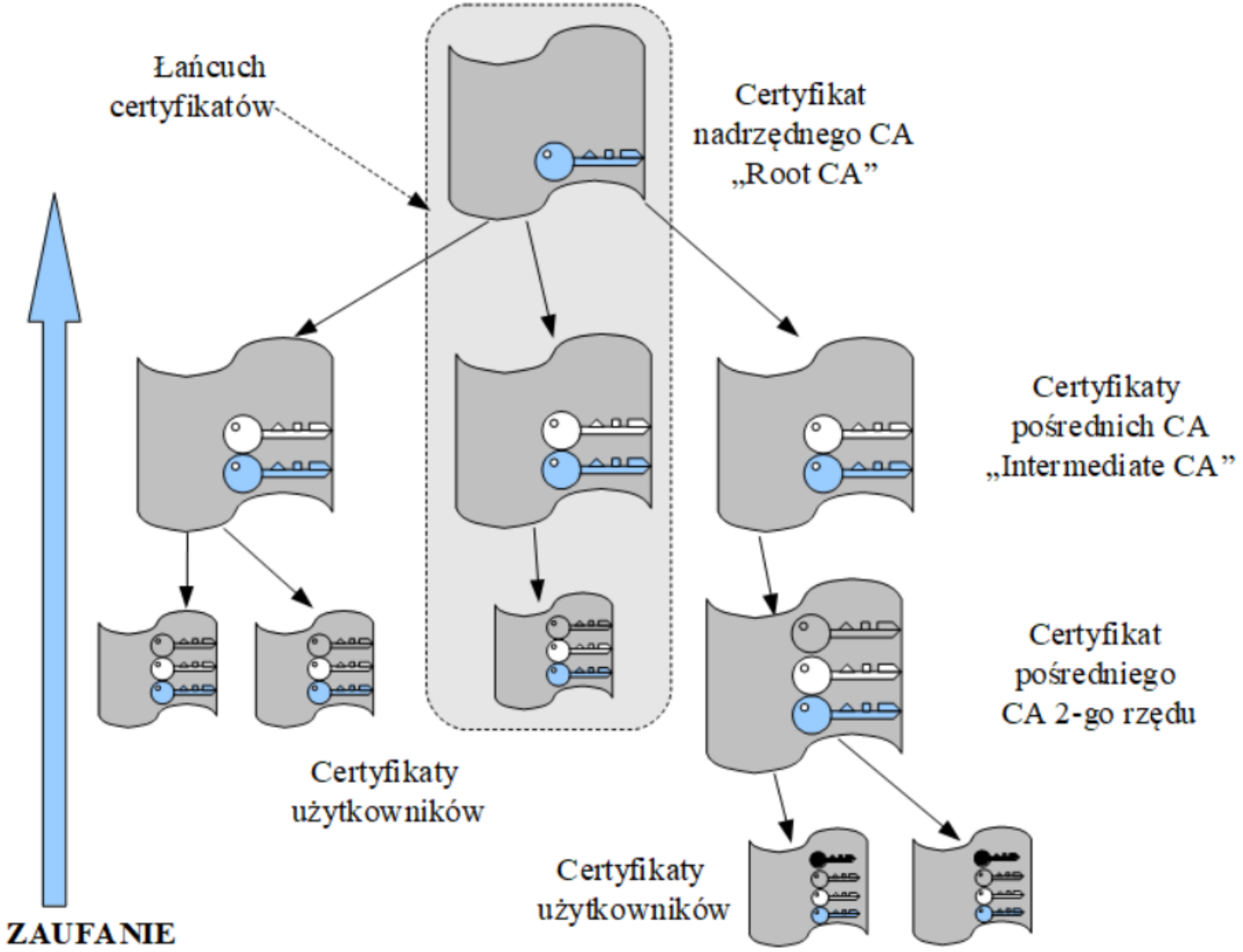
Certyfikaty

- Nie każdy certyfikat będzie uznany za godny zaufania.
- To czy użytkownik akceptuje dany certyfikat (tj. czy np. pozytywnie weryfikuje podpis cyfrowy złożony za pomocą tego certyfikatu) zależy także od polityki bezpieczeństwa w organizacji użytkownika.
- Ogólne reguły określające zaufanie do certyfikatu:
  - Certyfikat **musi** być poprawny (nieuszkodzony)
  - Certyfikat **musi** być ważny, tj. aktualny czas musi znajdować się w przedziale okresu ważności certyfikatu.
  - Certyfikat **nie może** znajdować się na liście certyfikatów unieważnionych.
  - Certyfikat **musi** być poprawnie **podpisany** przez wystawcę ***(lub samo-podpisany***)
- Główne reguły zależne od PBI:
  - Certyfikat samo-podpisany **może** być akceptowany
  - Zazwyczaj **wymaga** się istnienia **łańcucha zaufania,** prowadzącego od znanego CA do certyfikatu
  - W zastosowaniach konsumenckich zadawalamy się zazwyczaj listą CA „wbudowaną” w klienta (np. przeglądarkę WWW), jednak w organizacjach polityka akceptacji CA może być bardziej zróżnicowana.

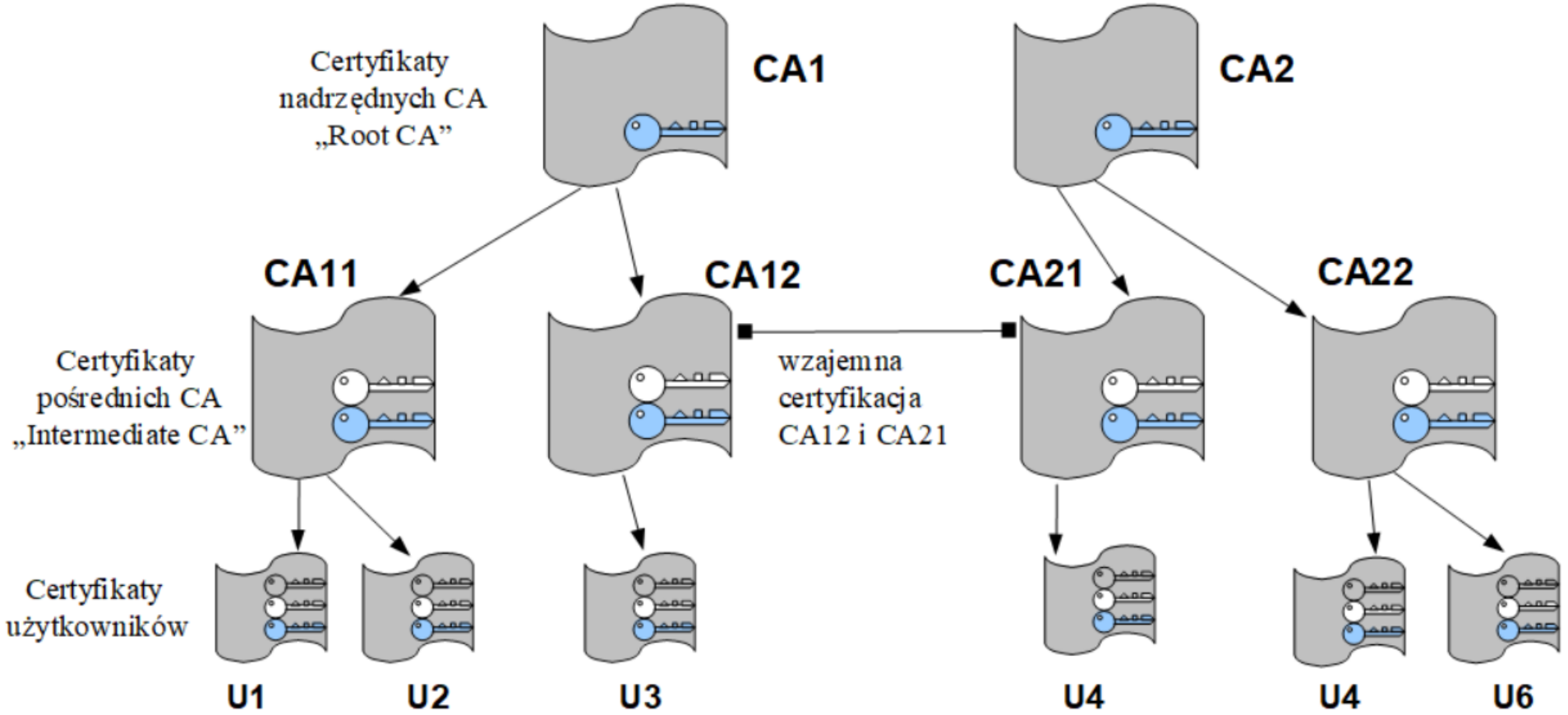
Certyfikaty i CA

- **Centrum Certyfikacji** lub Urząd Certyfikacji – **CA** (Certificate Authority) to jednostka organizacyjna obdarzona zaufaniem w zakresie tworzenia i wydawania użytkownikom certyfikatów klucza publicznego. Jest tzw. zaufaną „trzecią stroną”, która podpisując certyfikat użytkownika, uwiarygadnia go przed innym użytkownikiem.
- Głównym zadaniem CA jest wydawanie podpisanych przez nie certyfikatów. CA pełni także wiele innych funkcji organizacyjnych, takich jak: przyjmowanie wniosków o certyfikaty, prowadzenie ich rejestru, publikowanie list unieważnionych certyfikatów, itp.
- **Ścieżka certyfikatów** (scieżka certyfikacji) – uporządkowany ciąg certyfikatów, prowadzący od znanego zaufanego certyfikatu CA, wybranego przez weryfikującego, aż do weryfikowanego certyfikatu, utworzony w celu weryfikacji certyfikatu. Ścieżka certyfikatów spełnia następujące warunki:
  - Cert(n) jest weryfikowanym certyfikatem,
  - dla każdego certyfikatu Cert(i) należącego do ścieżki certyfikatów {Cert(1), Cert(2), ..., Cert(n-1)} podmiot certyfikatu Cert(i) jest wydawcą certyfikatu Cert(i+1),
  - Certyfikat Cert(1) jest wydany przez urząd certyfikacji, któremu ufa weryfikator.

Certyfikaty i CA



Certyfikaty i CA



TLS - wydajność

- 2 -10 razy wolniejsze niż samo TCP
- Narzut czasowy związany z:
  - fazą zestawiania sesji (operacje klucza publicznego po stronie klienta i serwera)
  - kryptografią symetryczną w trakcie sesji
- Z drugiej strony ...
  - pełna kompresja danych
  - zob. SPDY/HTTP 2.0 - dalej

TLS API Python

- Pakiet **ssl**
- Krytyczne dla bezpieczeństwa jest zrozumienie, co biblioteka zrobi automatycznie, a o co musi zadbać programista:
  - Weryfikacja serwera nie zawsze będzie automatyczna
  - Biblioteka nie zaakceptuje certyfikatu uszkodzonego lub nieważnego, ale nie zawsze zweryfikuje zgodność nazwy oraz łańcuch certyfikacji!
- Dla API Python o sposobie weryfikacji certyfikatu decyduje sposób zainicjowania **kontekstu** TLS
- Gniazd TLS (SSL) używamy wiążąc standardowo utworzone gniazdo z tzw. **kontekstem** TLS

TLS API Python

```
import socket
import ssl

hostname = 'www.ii.pw.edu.pl'
context = ssl.create_default_context()
with socket.create_connection((hostname, 443)) as sock:
    with context.wrap_socket(sock, server_hostname=hostname) as ssock:
        print(ssock.version())
```

- Tworzymy domyślny kontekst TLS, pokazana standardowa metoda dla klienta jest zazwyczaj najlepsza: ładuje certyfikaty CA, włącza walidację certyfikatów oraz zgodność nazwy serwera, dobierze też najlepszy zestaw algorytmów kryptograficznych
- Tworzymy gniazdo i łączymy się (TCP) z serwerem
- Następnie tworzymy sesję TLS związaną z tym gniazdem poprzez dowiązanie kontekstu – **uwaga**! - “owinięte” gniazdo może się zachowywać nieco inaczej od “zwykłego”!

TLS API Python – klient c.d.

```
# simple Python TLS socket client
# (c) G.Blinowski for PSI 2021


import socket
import ssl
import sys


HOST = 'www.cert.pl'
port = 443
context = ssl.create_default_context()
with socket.create_connection((HOST, port)) as sock:
    with context.wrap_socket(sock, server_hostname=HOST) as ssock:
      print(ssock.version())
      cert = ssock.getpeercert()
      print( cert )
      ssock.sendall(b'GET / HTTP/1.1\r\nHost: www.cert.pl\r\n\r\n')


      # read and print data ...
```

TLS API Python – klient c.d.

```
gjb@maersk:~/Projects/PSI$ python3 tls_client1.py
Will connect to  www.cert.pl : 443
TLSv1.3
{'subject': ((('countryName', 'US'),), (('stateOrProvinceName',
   'California'),), (('localityName', 'San Francisco'),),
   (('organizationName', 'Cloudflare, Inc.'),), (('commonName',
   'cert.pl'),)), 'issuer': ((('countryName', 'US'),),
   (('organizationName', 'Cloudflare, Inc.'),), (('commonName',
   'Cloudflare Inc ECC CA-3'),)), 'version': 3, 'serialNumber':
   '0FA14A754C2792516CF2ED9533FAC635', 'notBefore': 'Apr 22 00:00:00
   2021 GMT', 'notAfter': 'Apr 21 23:59:59 2022 GMT', 'subjectAltName':
   (('DNS', 'www.incydent.cert.pl'), ('DNS', '*.cert.pl'), ('DNS',
   'cert.pl')), 'OCSP': ('http://ocsp.digicert.com',), 'caIssuers':
   ('http://cacerts.digicert.com/CloudflareIncECCCA-3.crt',),
   'crlDistributionPoints':
   ('http://crl3.digicert.com/CloudflareIncECCCA-3.crl',
   'http://crl4.digicert.com/CloudflareIncECCCA-3.crl')}
Received b'HTTP/1.1 200 OK\r\nDate: Wed, 01 Dec 2021 14:14:11 GMT\r\
   nContent-Type: text/html\r\n ...
```

TLS API Python serwer

```
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain('/root/certchain.pem', '/root/private.key')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0) as sock:
    sock.bind((servername, 443))
    sock.listen(5)
    with context.wrap_socket(sock, server_side=True) as ssock:
        conn, addr = ssock.accept()
        ...
```

- Tworzymy własny kontekst TLS poprzez konstruktor **SSLContext**
- Uwaga: to serwer narzuca zgodność wersji protokołu
- Serwer musi posiadać certyfikat!

client / **server**	**SSLv2**	**SSLv3**	**TLS** [3]	**TLSv1**	**TLSv1.1**	**TLSv1.2**
SSLv2	yes	no	no [1]	no	no	no
SSLv3	no	yes	no [2]	no	no	no
TLS (SSLv23) [3]	no [1]	no [2]	yes	yes	yes	yes
TLSv1	no	no	yes	yes	no	no
TLSv1.1	no	no	yes	no	yes	no
TLSv1.2	no	no	yes	no	no	yes

TLS API Python c.d.

```
SSLContext.load_cert_chain(certfile, keyfile=None, password=None)
```

- **load_cert_chain()** ładuje arbitralny zestaw certyfikatów i kluczy z pliku
- Klucz prywatny może, ale nie musi być zaszyfrowany
- Argument *password* odkodowuje klucz prywatny - może być funkcją

```
SSLContext.wrap_socket(sock, server_side=False,
  do_handshake_on_connect=True, server_hostname=None, session=None)
```

- **Uwaga** na rozróżnienie gniazda: klienckie lub serwerowe
- Kontekst można dołączyć do niepołączonego gniazda klienckiego
- Parametr *server_hostname* pozwala na weryfikację nazwy i obsłużenie serwerów “różnych nazw” na jednym porcie
- **do_handshake..** - automatyczna lub ręczna inicjalizacja sesji TLS

TLS API Python c.d.

```
ssl.match_hostname(cert, hostname)
```

- Reguły zgodności określa: RFC 6125 – uwzględniają różne pola certyfikatu i różne warunki zgodności
- Hostname może zawierać * (ale są ograniczenia), itd.
- Zobacz: SSLSocket.getpeercert())

```
ssl.get_server_certificate(addr, ssl_version=PROTOCOL_TLS_CLIENT,
  ca_certs=None[, timeout])
```

- Sprawdza certyfikat dla abitralnego adresu serwera
- Weryfikuje certyfikat i łańcuch certyfikatów

TLS API C - przykład

```
#include <openssl/bio.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <openssl/x509.h>
#include <openssl/x509_vfy.h>
```

```
 char  dest_url[] =
"https://www.xyz.com";
  BIO  *certbio = NULL;
  BIO  *outbio = NULL;
  X509 cert = NULL;
  X509_NAME *certname = NULL;
  const SSL_METHOD *method;
  SSL_CTX *ctx;
  SSL *ssl;
  int server = 0;
```

```
/* Inicjalizacja biblioteki openssl */
  OpenSSL_add_all_algorithms();
   ERR_load_BIO_strings();
   ERR_load_crypto_strings();
   SSL_load_error_strings();
  if(SSL_library_init() < 0)
    BIO_printf(outbio, "Could't initialize OpenSSL!\n");
/* Obsługujemy SSL2/3 oraz TLS */
  method = SSLv23_client_method();
```

TLS API C - przykład

```
/* Tworzenie kontekstu */
   if ( (ctx = SSL_CTX_new(method)) == NULL)
      BIO_printf(outbio, "Unable to create new context\n");

/* Tworzenie obiektu połączenia SSL */
   ssl = SSL_new(ctx);

/* Połączenie z serwerem, create_socket() tworzy i łączy gniazdo
   TCP w standardowy sposób */

  server = create_socket(dest_url, outbio);

  if ( server != 0)
     BIO_printf(outbio,"TCP connection to: %s OK.\n", dest_url);

/* Powiąż kontekst  SSL z deskryptorem gniazda */
   SSL_set_fd(ssl, server);

/* Otwieramy sesję SSL */
   if ( SSL_connect(ssl) != 1 )
     BIO_printf(outbio, "Error: Couldn't build a SSL session\n”);
    else
     BIO_printf(outbio, "Successfully enabled SSL/TLS session to:
   %s.\n", dest_url);
```

TLS API C - przykład

```
/* Pobierz dane certyfikatu serwera */
    cert = SSL_get_peer_certificate(ssl);
    if (cert == NULL)
      BIO_printf(outbio, "Error: Could not get a certificate\n");
    else
      BIO_printf(outbio, "Retrieved the server's certificate
   from: %s.\n", dest_url);

/* Przykład pobrania składowych certyfikatu */

   certname = X509_NAME_new();
   certname = X509_get_subject_name(cert);

   BIO_printf(outbio, "Certificate subject data:\n");
       X509_NAME_print_ex(outbio, certname, 0, 0);


/* Zwalnianie struktur */
   SSL_free(ssl);
   close(server);
   X509_free(cert);
   SSL_CTX_free(ctx);
```

Certyfikat self signed (autopodpisany)

- Narzędzia do generowania certyfikatów: openssl, xca, OpenCA, narzędzia wchodzące w skład Microsoft 20xx Server, narzędzia AWS i wiele innych
- Gdy potrzebny jest certyfikat do eksperymentów i nie chcemy tworzyć hierarchii CA, możemy zastosować certyfikat **autopodpisany**, np. przy pomocy program openssl:

```
gjb@maersk:~$ openssl req -x509 -out certfile.pem -keyout keyfile.key
Generating a RSA private key
.....+++++.....+++++
writing new private key to 'keyfile.key'
Enter PEM pass phrase: ******
Verifying - Enter PEM pass phrase: ******
You are about to be asked to enter information that will be incorporated
   into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
   DN.
```

Certyfikat self signed (autopodpisany)

```
You are about to be asked to enter information that will be
incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a
DN.

-----

Country Name (2 letter code) [AU]:PL

State or Province Name (full name) [Some-State]:Mazowieckie

Locality Name (eg, city) []:Warszawa

Organization Name (eg, company) [Internet Widgits Pty Ltd]:IIPW

Organizational Unit Name (eg, section) []:ZOAK

Common Name (e.g. server FQDN or YOUR name) []:maersk

Email Address []:grzegorz.blinowski@pw.edu.pl

gjb@maersk:~$ ls -l

total 96

-rw-rw-r-- 1 gjb gjb  1062 Dec  7 15:14 certfile.pem

-rw------- 1 gjb gjb  1854 Dec  7 15:13 keyfile.key
```

Pliki typu .pem mają format tekstowy i mogą być używane przez wiele narzędzi PKI

Certyfikat self signed (autopodpisany)

- Inny przykład generowania certyfikatu – certyfikat ważny przez rok, ograniczony do silnych alogyrtmów kryptograficznych:

```
gjb@maersk:~$  openssl req -x509 -sha256 -nodes -days 365 -newkey
   rsa:2048 -keyout bettercert.key -out bettercert.pem
```

- Weryfikacja i podgląd danych certyfikatu:

```
gjb@maersk:~$ openssl x509 -in certfile.pem -text

Certificate:
       Data:
           Version: 3 (0x2)
           Serial Number:
               6a:69:c6:df:ff:63:ce:f1:a1:56:a7:2d:3e:87:8e:c5:49:ae:1d:e0
           Signature Algorithm: sha256WithRSAEncryption
           Issuer: C = PL, ST = Mazow, L = Warszawa, O = IIPW, OU = ZOAK, CN =
   maersk, emailAddress = g.blinowski@ii.pw.edu.pl
           Validity
               Not Before: Dec  7 15:50:25 2021 GMT
               Not After : Jan  6 15:50:25 2022 GMT
           Subject: C = PL, ST = Mazow, L = Warszawa, O = IIPW, OU = ZOAK, CN
   = maersk, emailAddress = g.blinowski@ii.pw.edu.pl
           … … … …
```

Materiały dodatkowe TLS

- Python - opis biblioteki ssl:
  - https://docs.python.org/3/library/ssl.html
- Należy zapoznać się zwłaszcza z sekcją:
  - https://docs.python.org/3/library/ssl.html#ssl-security

- Przykład dla języka C:
  - https://fm4dd.com/openssl/sslconnect.shtm

- Biblioteka OpenSSL
  - https://www.openssl.org/
- XCA – proste narzędzie do obsługi certyfikatów X509v3
  - https://hohnstaedt.de/xca/

HTTP 2.0 i 3.0

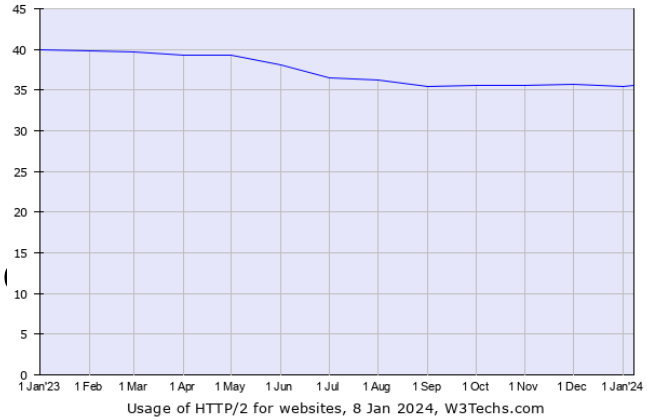
Co dalej z HTTP ?

- Protokół HTTP ma ponad 30 lat, 30 lat temu:
  - łącze abonenckie ma 56 – 128 Kbit/s
  - klient jest jednowątkowy, strona uboga w grafikę zajmuje kilka-kiladziesiąt KB
- Obecnie przeciętna strona (Web 2.0):
  - ~600 KB, w tym: skrypty, style, kilkadziesiąt osadzonych obiektów generujących osobne wywołania HTTP
- Negocjacja sesji TCP (3-way hanshake) jest kosztowna czasowo
- Mechanizm TCP “slow start” dodatkowo opóźnia transmisję
- Keep-alive tylko częściowo rozwiązuje problem bo:
  - nagłówki są powielane (powtarzane), nagłówki są duże: do 2 KB (głównie cookie), powolne zlecenie może blokować następne w kolejce
- Dodatkowo:
  - kompresja nie obejmuje nagłówków
  - strona generuje maks do 6 sesji TCP na serwer
  - “Domain sharding”

HTTP 2.0 – główne cechy

RFC 7440
RFC 7441

- Protokół binarny (a nie tekstowy, jak HTTP)
- W pełni multipleksowany – w sesji TCP HTTP/2 wiele niezależnych dwukierunkowych “pod strumieni” danych
  - Eliminuje konieczność zestawiania równoległych połączeń TCP do tego samego serwera
- Jedna w całości binarna, kompresowana sesja:
  - nagłówki też podlegają kompresji
  - redundantne nagłówki nie są powielane (user-agent, accept-*, itp.) - do 88% oszczędności (czasami...)
- Sesja pozwala na równoległe żądania HTTP
- Opcja “push” - od serwera do klienta
- Szyfrowanie TLS niewymagane (ale wskazan
- Nie jest wstecznie zgodne z HTTP/1.1
- Obecnie (12/2023) ~ 36% serwisów obsługuje HTTP/2



https://w3techs.com/technologies/details/ce-http2

Przełączenie na HTTP/2

- Połączenie HTTP/2 może być natychmiastowe lub negocjowane:
  - Rozpoczynamy “zwykłe” połączenie HTTP/1.1
  - Przy pomocy nagłówka “upgrade” próbujemy przełączyć się na HTTP/2.0

K:

```
GET /default.htm HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: HTTP/2.0
HTTP2-Settings: ...
```

S:

```
HTTP/1.1 200 OK
Content-Length: 243
Content-Type: text/html
```

Serwer nie obsługuje HTTP 2, upgrade ignorowane

S:

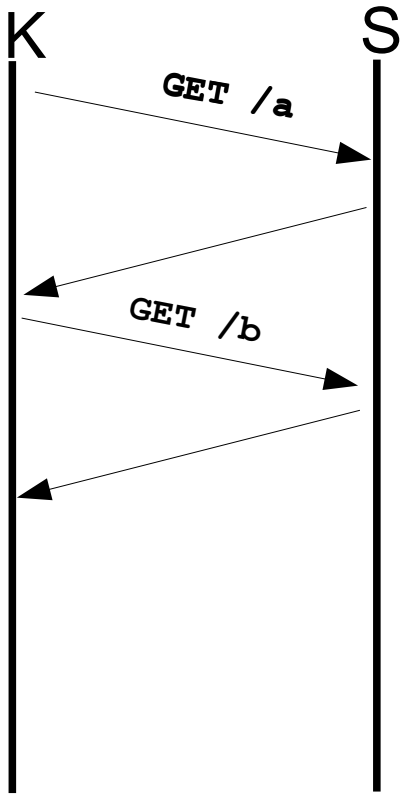
```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: HTTP/2.0

[ HTTP/2.0 connection …
```

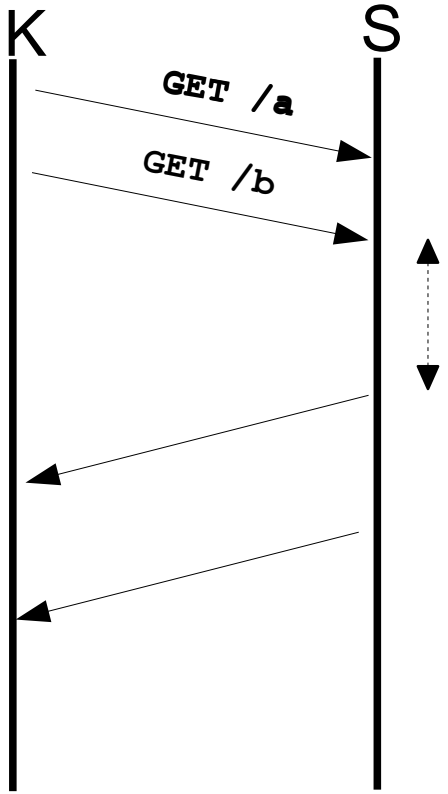
Serwer obsługuje HTTP 2

Przełączenie na HTTP 2

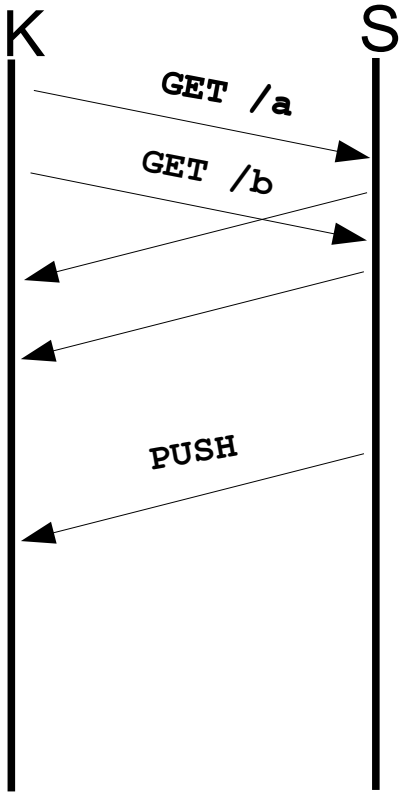
SPDY / HTTP/2 – mutliplexing



HTTP
keepalive

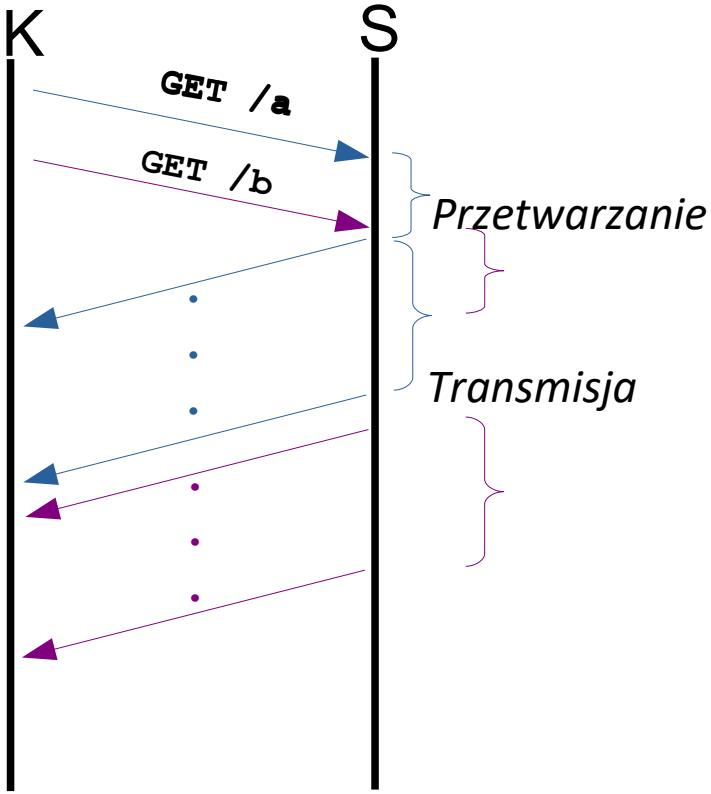


HTTP 1.1 keepalive +
pipelining

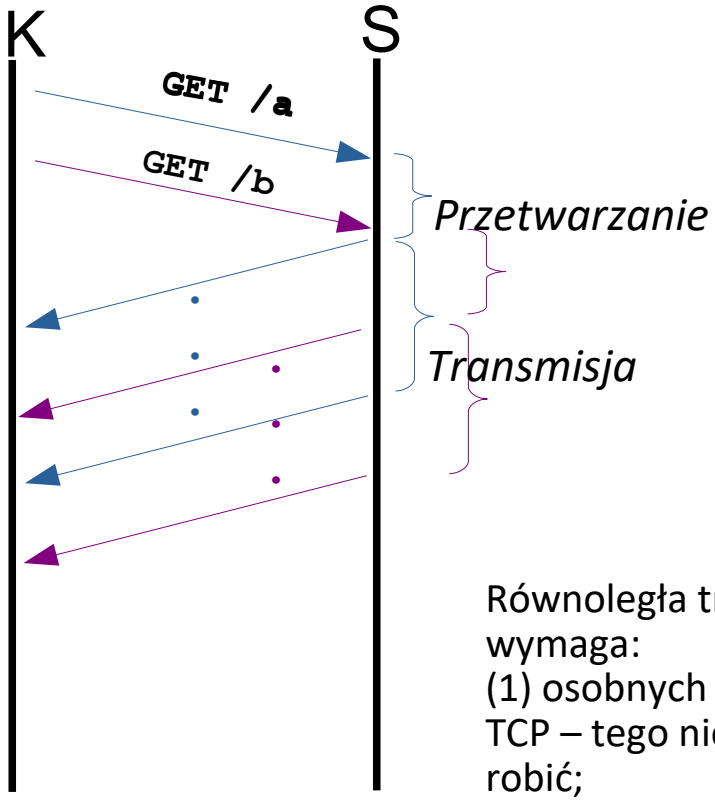


SPDY, HTTP/2
multiplexing

head-of-line (HOL) blocking w HTTP



Pipelining

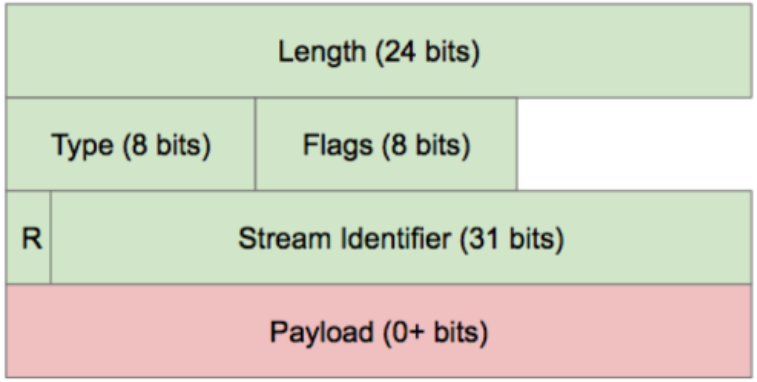


Multiplexing

Równoległa transmisja wymaga:
(1) osobnych połączeń TCP – tego nie chcemy robić;
(2) podziału danych na małe porcje (ramki / frames)

HTTP2 FRAME (ramki)

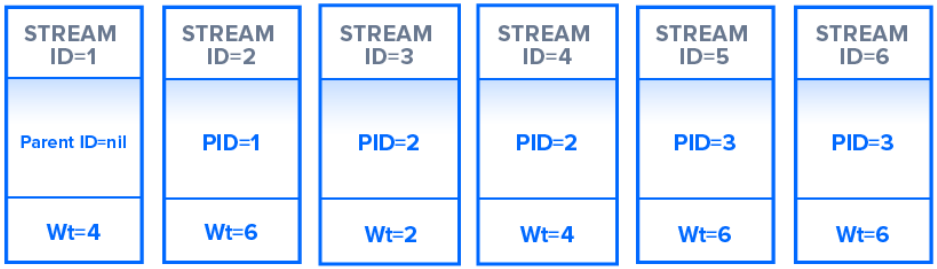
- Klient wysyła do serwera nagłówek połączenia:
  `PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n`
- Następnie przesyłane są binarne ramki (frames), ramka jest najmniejszą podstawową jednostką komunikacji
- Zastosowanie ramek pozwala na strukturalizację komunikacji i stosowanie wielu równoległych sesji (multipleksowanie)
- Typy ramek: nagłówki; dane; dane sterujące (np. sterowanie przepływem)
- Wiadomość (messsage): request / response
- Wiadomości przesyłane są w strumieniach
- Wiele strumieni w jednej sesji TCP
- Strumienie są dwukierunkowe
- Strumienie tworzą hierarchię



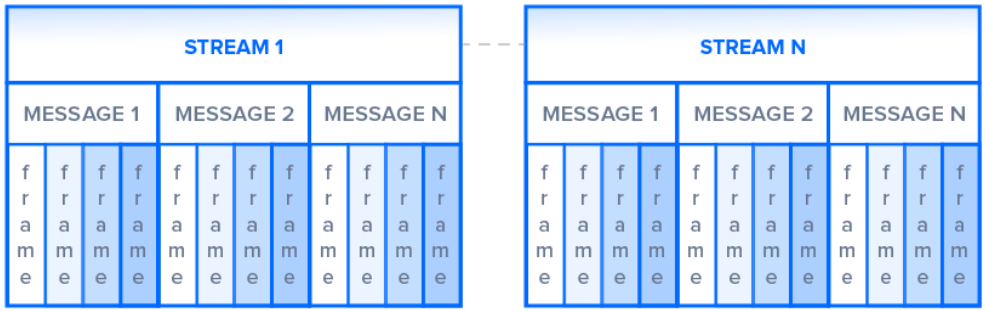
- Sterowanie przepływem w obrębie strumienia – w TCP działa sterowanie przepływem, ale w HTTP2 chcemy mieć podobny mechanizm dla każdego strumienia wewnątrz jednej sesji TCP!
- Dostępna jest też komunikacja typu “push” inicjowana od serwera do klienta - (“wypychanie” zasobów, które będą klientowi potrzebne)

Frames, streams (ramki, strumienie)

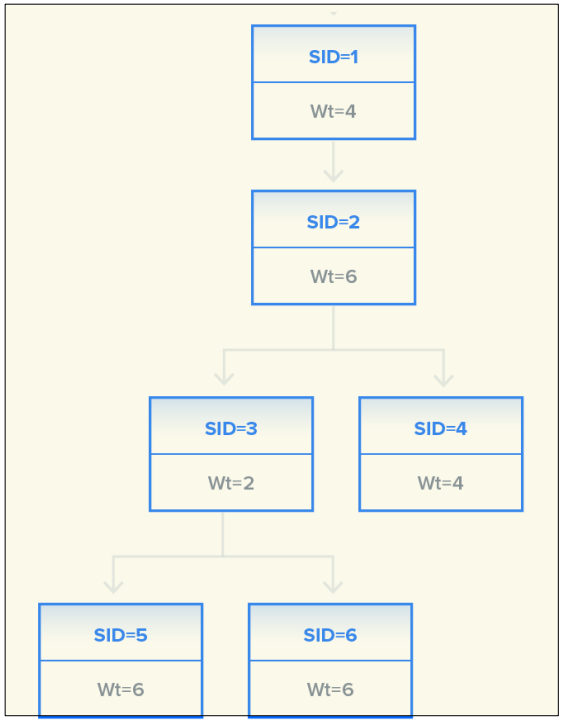
- Podział na binarne ramki pozwala na realizację wielu równoległch transmisji.
- Odpowiada to logicznie wielu sesjom TCP (przeglądarka nie może otworzyć zbyt wielu sesji do serwera)
- Ramki w poszczególnych strumieniach mogą się przeplatać, co usprawnia przepływ i zapobiega sytuacji, gdy jedno wolniej obsługiwane żądanie blokuje pozostałe.



każdy strumień posiada rodzica (PID) oraz wagę (priorytet)



Jeden strumień TCP (HTTP lub HTTPS)



Kompresja nagłówków

- Tylko pierwsze żądanie/odpowiedź w sesji przesyła pełne nagłówki
- Później przesyłane tylko te nagłówki, które ulegają zmianie
- SPDY – kompresja gzip, ale atak CRIME, HTTP/2 – alg. Huffmana
- Stosowana jest kompresja ze statyczną tablicą najczęściej występujących nagłówków (toknenów) i kodowaniem Huffmana dla pozostałych napisów

HTTP/1.1

```
GET /index.html HTTP/1.1
Host: www.foo.com
Referer:http://www.example.com/
Accept-Encoding:gzip
```

```
GET /logo.jpg HTTP/1.1
Host: www.foo.com
Referer:http://www.f.com/i.html
Accept-Encoding:gzip
```

HTTP/2.0

```
:method: GET
:scheme: http
:host: www.foo.com
:path: /index.html
referer: http://www.example.com/
accept-encoding: gzip
```

```
:path: /logo.jpg
referer: http://www.f.com/i.html
```

HTTP 3.0

- RFC 9114; zaprojektowany i zaimplementowany 2018 - 2021
- Zgodny z HTTP 2.0; obsługiwany obecnie (2022) przez około 75% zainstalowanych przeglądarek
- Różnica w stosunku do HTTP 1.1 i 2.0 w transporcie:
  - HTTP **2.0** bazuje na TCP,
  - HTTP **3.0** wykorzystuje QUIC (zob. W3 - prot. transportowe);
  - Przypomienie: protokół QUIC Nazywany „TCP/2”, dostosowany do obsługi HTTP/3, zoptymalizowany do transmisji wielu równoległych strumieni danych w HTTP, eliminuje wady wydajnościowe TCP

Klienckie biblioteki i narzędzia obsługujące HTTP

- C/C++:
  - wget – uniwersalny klient command line HTTP(s), FTP; GNU
  - CURL (client URL) – klient i biblioteka
    - Przenośne API dla kilkudziesięciu języków programowania
    - Obsługuje: HTTP(S), FTP i kilka innych protokołów; metody: GET, HEAD, POST, PUT, DELETE (model REST)
    - Obsługuje SSL i certyfikaty
    - C - biblioteka libcurl

```
#include <curl/curl.h>

   CURL *curl;
   CURLcode res;


   curl = curl_easy_init();
   curl_easy_setopt(handle, CURLOPT_URL, http://www.ii.pw.edu.pl/");
   res = curl_easy_perform(curl);

   curl_easy_cleanup(curl);
```

CURL c.d.

- Dane w CURL pobiera się i przekazuje przy pomocy funkcji callback
- Dodatkowe opcje (np. nagłówki) ustawia się przy pomocy curl_easy_setopt()
- Dane zwrotne przekazuje się przez callback

```
#include <curl/curl.h>

CURL *curl;
CURLcode res;

curl = curl_easy_init();
curl_easy_setopt(handle, CURLOPT_URL, "http://www.ii.pw.edu.pl/");

/* ustawmy nagłówek user-agent */
curl_easy_setopt(curl_handle, CURLOPT_USERAGENT, "libcurl-agent/1.0");

/*  chcemy wczytać odebrane dane do bufora */
curl_easy_setopt(curl_handle, CURLOPT_WRITEFUNCTION, WriteMemoryCallback);
res = curl_easy_perform(curl);

static size_t
WriteMemoryCallback(void *contents, size_t size, size_t nmemb, void *userp)
/* zob: https://curl.se/libcurl/c/getinmemory.html */
```

Python httplib, urllib

- httplib – biblioteka “niskopoziomowa”
- urllib - wygodna biblioteka obsługująca łączność HTTP

```
urllib.request.urlopen(url, data=None, [timeout, ]*,  cert-params )
```

- Przykład:

```
import urllib.request

#  zwróć pierwsze 1024B pobrane z danego URL
   with urllib.request.urlopen('http://www.python.org/') as f:
      print( f.read(1024) )

# dodaj opcje HTTP do wywołania
   req = urllib.request.Request('http://www.example.com/')
   req.add_header('Referer', 'http://www.python.org/')
   req.add_header('User-Agent', 'urllib-example/1.0')
   r = urllib.request.urlopen(req)
```

Zasoby do API HTTP

- **C/C++:**
  - https://www.gnu.org/software/wget/
  - https://curl.se/libcurl/
  - https://developer.ibm.com/articles/what-is-curl-command/
  - https://curl.se/libcurl/c/example.html

- **Python**:
  - https://docs.python.org/2/library/httplib.html
  - https://docs.python.org/3/library/urllib.html
  - https://docs.python-requests.org/en/latest/

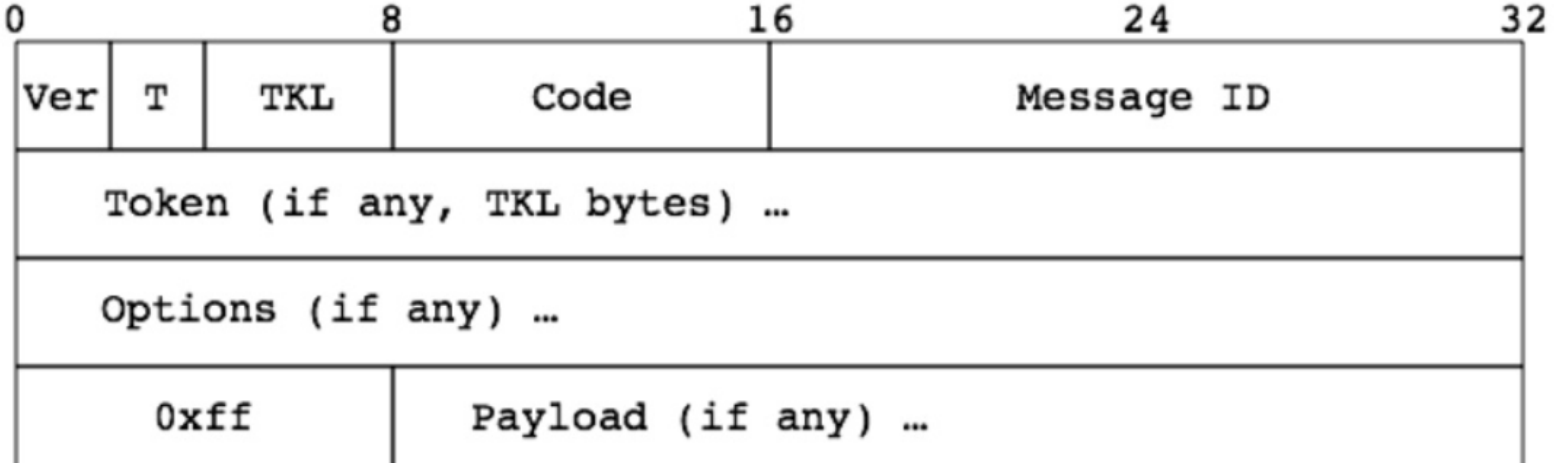
Systemy IoT
i
Protokół CoAP

Protokół CoAP

CoAP/1.0 - **RFC7252**

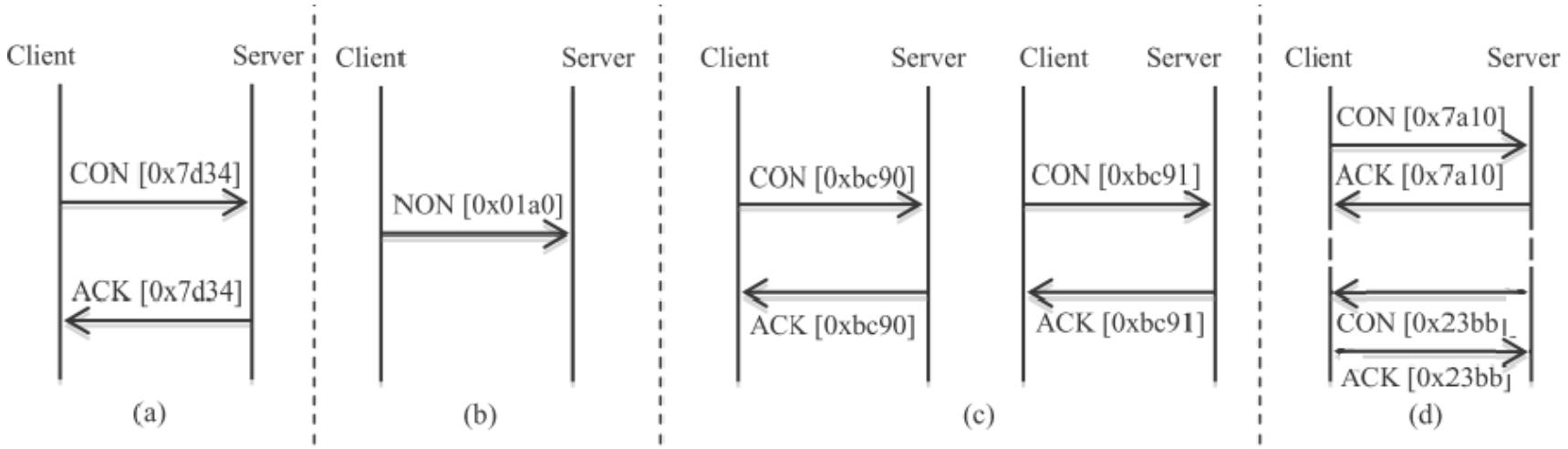
- Potrzeba prostego protokołu dla urządzeń IoT (Internet of Things): sensory, sterowniki, itp. - mocno ograniczone pod względem mocy CPU i wielkości RAM – HTTP1.1/2.0 zbyt skomplikowany
- CoAP: Constrained Application Protocol
  - "podobny" do HTTP: coap://example.com:5683/~sensors/temp.xml
  - Możliwość tłumaczenia CoAP na HTTP przez serwer proxy
  - Prostota: bazuje na UDP (unikamy sesji TCP, całość komunikacji w 1 datagramie zapytania oraz 1 datagramie odpowiedzi)
  - UDP pozwala też na rozgłaszanie zapytań *(np. polecenie wyłączenia oświetlenia powinno być rozgłoszone, a nie wysyłanie indywidualnie do wszystkich sterowników*)
  - TCP i DTLS obsługiwane opcjonalnie
  - Bazuje na modelu REST: polecenia GET, POST, PUT, DELETE: odpytanie, aktualizacja, tworzenie i usuwanie zasobu
  - RFC ma 112 stron, prosta implementacja (b. prosty parser)

Protokół CoAP c.d.



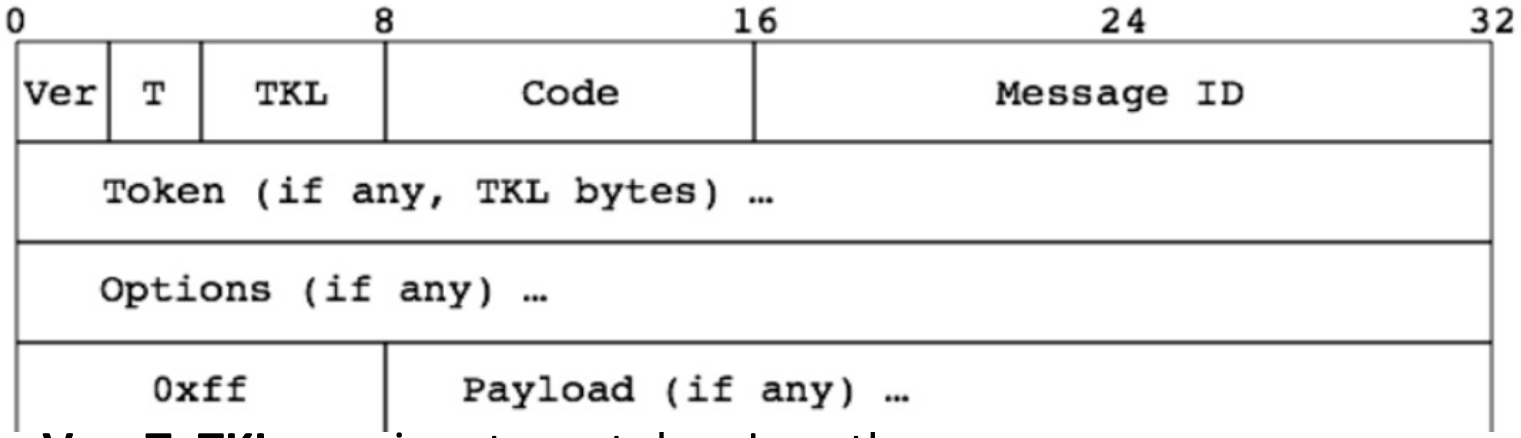
- Z uwagi na "kompaktowość" (jeden datagram UDP!) protokół jest binarny
- 4-y bajty nagłówka
- **Ver** - Obecnie "01"
- **T** – Message type (2b.): 0x1 – Confirmable; 0x1 – NonC; 0x2 – Ack; 0x3 – Reset (zob. następny slajd)
- Retransmisja z wykł. wycofywaniem,
  wykrywanie duplikatu na podstawie 16-bit **Message-ID**

Protokół CoAP c.d.



- Typy wiadomości I tryby komunikacji CoAP:
  - (a) potwierdzana (**CON** – confirmable); serwer odsyła **ACK** lub **RST**
  - (b) niepotwierdzana (**NON** - nonconfirmable)
  - (c) odpowiedź z danymi (piggyback response) – gdy serwer może odpowiedzieć “od razu”
  - (d) osobne potwierdzenie oraz dane (separate response) – gdy serwer będzie w stanie przesłać odpowiedź po pewnym czasie (jako osobny komunikat wymagający potwierdzenia)

Protokół CoAP c.d.



- **Ver, T, TKL:** version, type, token Length
- **Code** (8b.) Request/Response: ***class*** (3 b.), ***detail*** (5b.) - odpowiednik statusu 200, 404, etc.; zapisywany np. "2.05", "4.04", itd.
  - Class: 0 – Reqest; 2 - Success; 4 – Client Error; 5 – Server Error
- **Token – kontekst aplikacji - serwer odsyła token, aplikacja może dopasować odpowiedź do zapytania**
- **Opcje** – binarne, odp. nagłówków; zawierają m.in.: URI (Uri-Host, Uri-Port, Uri-Path, and Uri-Query), typ danych, max-age, Etag, itd.
- **Payload** (opcjonalny)
  - Format – zgodny z **Content-format** zapisanym w opcjach z ustaloną listą (text, json, xml, …)