Services and Applications Example: Web Browsing

Contents - Services and Applications - Web Browsing

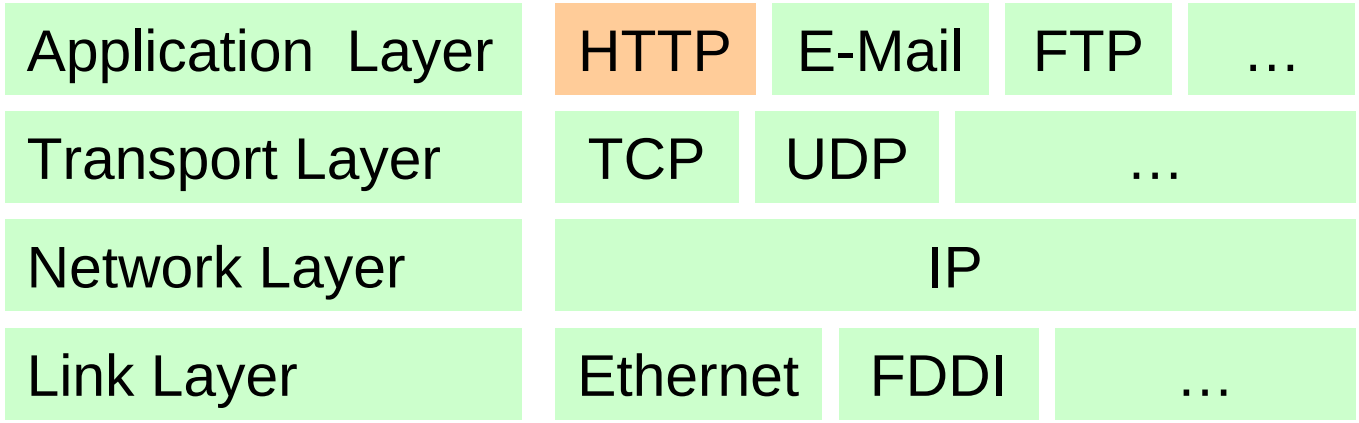
- Hypertext Transfer Protocol (HTTP)
- Addressing of Web Resources (URI, URL, URN)
- Web 2.0 Mashup
- Simple Object Access Protocol (SOAP)

Hypertext Transfer Protocol (HTTP)

Contents - HTTP

HTTP Introduction (1)

- Classification in the Internet Protocol Suite



- Invented by Tim Berners-Lee at CERN in 1990
- Major use in the beginning: retrieving interlinked resources (hypertext documents), establishment of the WWW

HTTP Introduction (2)

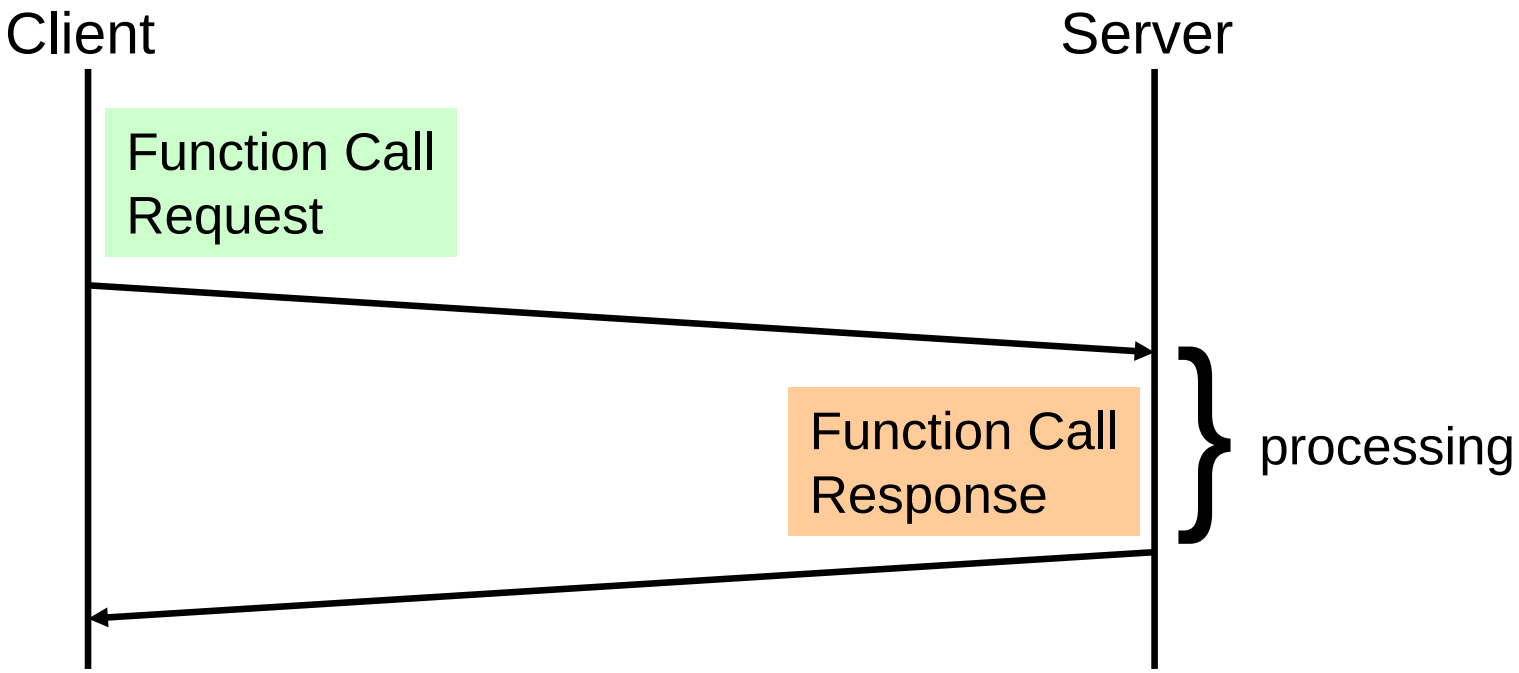
- **“The Hypertext Transfer Protocol (HTTP)** is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a **generic, stateless, protocol** which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through **extension** of its **request methods, error codes and headers**. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred. …” (RFC 2616, Abstract)
- HTTP is the principal mechanism for transferring web resources
- Actual IETF standard: RFC 2616 – Hypertext Transfer Protocol -- HTTP/1.1
- Development by the World Wide Web Consortium (W3C) and the IETF

HTTP Characteristics

- HTTP is used to **transfer** resources (e.g. web pages, pictures, java-script, other MIME type casted resources) between a user agent and a server following the request-response model
- Exchange of data is based on TCP/IP
- HTTP is a transfer and not a transport protocol
- Communication between user agent and server is stateless
- Two message types exist: request and response
- Messages are ASCII coded
- Messages are used to implement methods: GET, POST, HEAD, …

HTTP Characteristics - Request-Response-Model

- The Request-Response-Model is one of the message exchange patterns



- Synchronous: every operation determines a communication relation

HTTP Versions

- 1990 – HTTP/0.9
  - Only one method is provided: GET (no other approaches)
  - Not extensible, no support for versioning
- Ideas for versioning and robustness
  - Versioning for HTTP (RFC 2145, May 1997)
  - Robustness (backwards compatibility)
- May 1996 – HTTP/1.0 (RFC 1945)
  - Versioning
  - Stateless protocol
  - Support for extensions
  - Arbitrary content can be delivered
  - Today HTTP/1.0 is still common
- June 1999 – HTTP/1.1 (RFC 2616), extended by TLS (RFC 2817)
  - Enables persistent connections and the use of proxies
  - Many improvements reg. to performance: request pipelining, shared hosting
  - Extension for secure connections by Transport Layer Security TLS

Generic HTTP Message Format (RFC 822)

- The generic message format enables extensions
- This principle is used in all protocols which are based on HTTP
- HTTP-message = Request | Response
- Generic-message = Start-Line
  *(Message-Header)
  CRLF
  [Message-Body]
- Start-Line = Request-Line | Response-Line (Status-Line)
- Message-Header = Field-Name ":" [Field-Value] CRLF
- Field-Name = token
- Field-Value = *(Field-Content | LWS)
- Message-Body = Entity-Body | Entity-Body encoded as per Transfer-Coding
  - Presence signalled by header field Content-Length or Transfer-Encoding

HTTP Message Examples - HTTP-Request (simplified)

- Request = Request-Line
  *(Headers)
  CRLF
  [Message-Body]

- Request-Line = Method SP Request-URI SP HTTP-Version CRLF
- Method = “GET” | “POST” | “HEAD” | …
- HTTP-Version = “HTTP/1.0” | “HTTP/1.1” | …
- Headers = Name:Value
  - Name: name of the header
  - Value: value of the regarding header
- Message-Body = data (text)

HTTP Message Examples - HTTP-Request

```
GET / HTTP/1.0
```

```
GET / HTTP/1.1
Host: www.heise.de
User-Agent: Mozilla/5.0 (X11; U; Linux i686; de; …
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

HTTP Message Examples - HTTP-Response (simplified)

- Response = Status-Line
  *(Headers)
  CRLF
  [Message-Body]

- Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
- HTTP-version = “HTTP/1.0” | “HTTP/1.1” | …
- Status-Code = 3-digit integer result code
- Reason-Phrase = short textual description of the Status-Code
- Headers = Name:Value
  - Name: name of the header
  - Value: value of the regarding header
- Message-Body = data (text)

HTTP Message Examples - HTTP-Response

HTTP/1.0 200 OK
Date: Mon, 12 Oct 2009 08:21:30 GMT
Server: Apache/2.2.9
Content-Length: 12631
Content-Type: text/html; charset=utf-8

<HTML> Data … a lot of data </HTML>

HTTP/1.1 200 OK
Date: Mon, 12 Oct 2009 08:26:38 GMT
Server: Apache/2.2.9
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 13839
Connection: close
Content-Type: text/html; charset=utf-8

`{ƎG1#;E$┼ HFU"._Nj8稠 ...

Common HTTP Methods (1)

- HTTP methods are used in the context of HTTP-Requests
- GET
  - retrieve whatever information is identified by the Request-URI
- POST
  - is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line
  - POST is designed to allow a uniform method to cover the following functions:
    - Annotation of existing resources
    - Providing a block of data, such as the result of submitting a form, to a data-handling process
    - Extending a database through an append operation
- HEAD
  - identical to GET except that the server MUST NOT return a message-body in the response

Common HTTP Methods (2)

- OPTIONS
  - represents a request for information about the communication options available on the request/response chain identified by the Request-URI
- PUT
  - requests that the enclosed entity be stored under the supplied Request-URI
- DELETE
  - requests that the origin server delete the resource identified by the Request-URI
- TRACE
  - is used to invoke a remote, application-layer loop-back of the request message
  - the final recipient of the request SHOULD reflect the message received back to the client as the entity-body of a 200 (OK) response
  - is mainly used for the process of development

Common HTTP Headers (1)

- Content-Type
  - indicates the media type of the entity-body sent to the recipient
- Expires
  - gives the date/time after which the response is considered stale
  - Important for caching
- Host
  - specifies the Internet host and port number of the resource being requested
  - required for shared hosting
- Last-Modified
  - indicates the date and time at which the origin server believes the variant (object) was last modified
- User-Agent
  - contains information about the user agent originating the request
  - Important for personalization, internationalization, and optimization of web pages

Common HTTP Headers (2)

- Location
  - is used to redirect the recipient to a location other than the Request-URI for completion of the request or identification of a new resource
  - Important principle in several protocols which are based on HTTP (e.g. security)
  - Is used to implement so called “Redirects”
- Referrer
  - allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained
  - Useful for maintenance
    - From which web pages are the users coming?
    - Logging, optimization, caching, …
- Beyond that many other header fields exist, which can be used for different requirements

HTTP Status Codes and Reason Phrases

- 1xx: Informational – Request received, continuing process
- 2xx: Success – The action was successfully received, understood, and accepted
  - 200: OK
  - 201: Created – request fulfilled, new resource created
- 3xx: Redirection – Further action must be taken in order to complete the request
  - 301: Moved Permanently
  - 302: Moved Temporarily
- 4xx: Client Error – The request contains bad syntax or cannot be fulfilled
  - 400: Bad Request – request not understood
  - 401: Unauthorized – request requires user authentication
  - 403: Forbidden – server does not wish to make this information available
  - 404: Not Found – server has not found anything matching the Request-URI
- 5xx: Server Error – The server failed to fulfil an apparently valid request

Conclusions

- HTTP is pretty simple
- Actual Standard is HTTP/1.1
- There are many extensions for HTTP: HTTPS, Cookies, WebDAV, …
- Many protocols are based on HTTP: SOAP, XML-RPC, WS*, …

Practical Exercises

- Exercises:
  - Visit the following web pages:
    - http://www.brumaservice.com
    - http://www.sabine-andrae.com
    - http://www.paycunia.net
    - http://www.sepa-lastschrift.com
    - http://www.kreditkartenakzeptanz.net
  - Do they have something in common?
  - Ping them! What do you recognize? Can you explain it?
  - How is the concept called regarding to your observations?

Addressing of Web Resources (URI, URL, URN)

Contents - Addressing of Web Resources

Introduction

- Addressing of a service is determined by the protocol
  - IP via IP address
  - TCP/UDP via port numbers
- Requirements for names/addressing of an endpoint
  - Identification of arbitrary resources (web page, operation, function, file, email, …)
  - Support for arbitrary transport and transfer protocols (TCP/IP, UDP/IP, but also application layer protocols like HTTP and beyond)
  - Relative addressing of resources to actual context
  - Extensible
  - Simple spelling (ASCII, 7 bit)
  - Readable

Definitions

- Uniform Resource Identifier (URI)
  - Actual IETF standard: RFC 3986
  - Generic expression for addressing of arbitrary resources
  - Subsets of URIs are URLs and URNs
- Internationalized Resource Identifier (IRI)
  - Complement to URIs
  - IRI is a sequence of characters from the Universal Character Set (Unicode)
  - Mapping from IRI to URI is defined
- Uniform Resource Locator (URL)
  - Subset of URI-schemes
  - “URLs provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location")” (RFC 3986)
- Uniform Resource Name (URN)
  - URI with institutional aspect
  - Its used for providing persistent names for resources

Uniform Resource Identifier (URI)

- URI definition (RFC 3986)
  - Generic expression for addressing of arbitrary resources

URI - Syntax

- URI – syntax for identifier

  <uri> ::= <scheme>”:”<scheme-specific-part> **(RFC 2369)**

  <uri> ::= <scheme>”:”<hier-part> [”?” query] [”#” fragment] **(RFC 3986)**

- <scheme>
  - Name of the URI scheme
- <scheme-specific-part>
  - Identifier regarding to the URI scheme

URI - Coding Rules

- URIs are coded in ASCII
  - All characters are allowed but some special rules have to be followed:
    - Percentage sign (“%”, ASCII 25h)
      - Function: escape sign
    - Hierarchical Mapping (“/”, ASCII 2Fh)
      - Separation of strings via “/” implies hierarchy
    - Fragment delimiter (“#”, ASCII 23h)
      - Identifies a fragment within a resource
    - Query delimiter (“?”, ASCII 3Fh)
      - Separation in resource specific part and query specific part of an URI

Uniform Resource Locator (URL)

- URI – syntax for identifier

  <uri> ::= <scheme>”:”<scheme-specific-part>
- URL scheme definition (RFC 1738)
  - Information about the access is provided

  <scheme> ::= “http” | “https” | “ftp” | “news” | “mailto” | “nntp” | …
- URL scheme specific part

  <scheme-specific-part> ::=
  [“//”] [user [“:” password] “@”] host [“:” port] [“/” url_path]
  - Definitions are assigned by the Internet Assigned Numbers Authority (IANA)

URL - Example: HTTP Scheme

- URI – syntax for identifier

  <uri> ::= <scheme>”:”<scheme-specific-part>

- HTTP URL

  <scheme> ::= “http”

  <scheme-specific-part> ::=

  [“//”] [user [“:” password] “@”] host [“:” port] [“/” abs_path]

  - **<host>: legal internet host domain name or IP address**
  - **<port>: *DIGIT**
  - **<abs_path> ::=**

    “/”[<path>][“;” <params>][“?” <query>][“#” <fragments]
  - Examples
    - http://www.ngi-lecture.edu/makes/fun?answer=yes#itdoes
    - http://userid:pwd@sparschwein.de/howheavyareyou?period=lastmonth#FragId

Uniform Resource Name (URN)

- URN scheme definition (RFC 1737, 2141)
  - “Uniform Resource Names (URNs) are intended to serve as persistent, location-independent, resource identifiers and are designed to make it easy to map other namespaces (which share the properties of URNs) into URN-space. Therefore, the URN syntax provides a means to encode character data in a form that can be sent in existing protocols, transcribed on most keyboards, etc.” (Introduction RFC 2141)
- URN

  <scheme> ::= “urn”

  <scheme-specific-part> ::= <nid> “:” <nss>
  - **<nid> = namespace identifier**
  - **<nss> = namespace specific string**
  - Examples
    - urn:isbn:0123456789
    - urn:ietf:rfc:2141

URN - Attributes

- URN attributes
  - Global focus
  - Unique
  - Persistent
  - Scalable
  - Extensible
  - Resolvable
  - …

URL vs. URN

	URN	URL
Scope	Global	Global (abs. URL) Local (rel. URL)
Globally Unique	Yes	Yes (abs. URL) No (rel. URL)
Persistent	Yes	No
Scalable	Yes	Yes
Legacy Support	Yes	Limited
Resolution	Not yet determined	Partly using DNS

Web 2.0 Mashup

Contents - Web 2.0 Mashup

- What is a Mashup?
- Mashup Mode of Operation
- Mashup Measurements
- Conclusions

What is a Mashup?

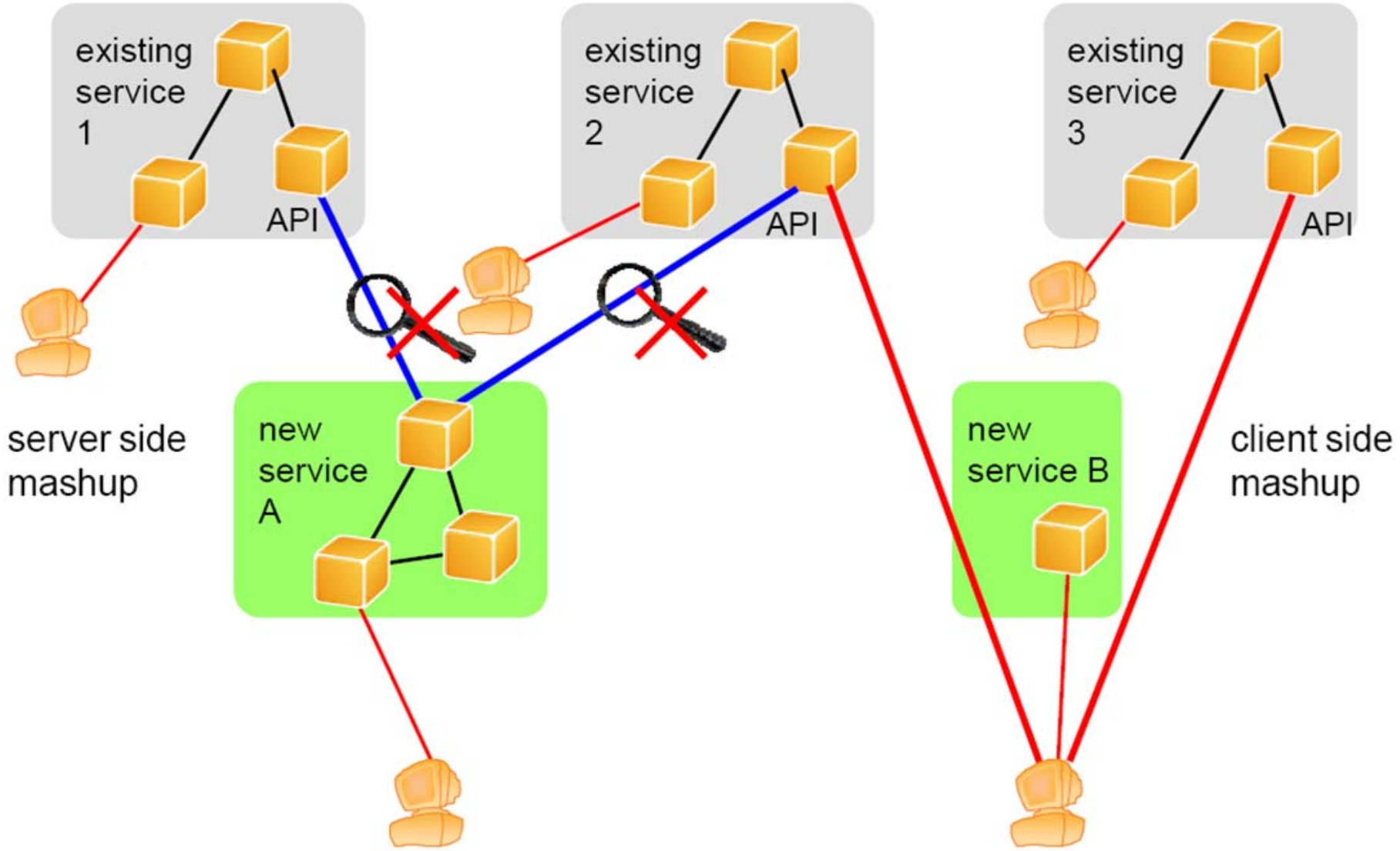
- A Mashup is a web page or application that uses and combines data, presentation or functionality from two or more sources to create new services.
  (Wikipedia)

→ Polished up web interfaces to combine, better visualize or aggregate already available information or services

→ “re-mix” of digital data

Mashup Mode of Operation

- “re-mix” of digital data



Mashup Mode of Operation

- "Web Inclusion" vs. "Mashup"

"Classical" Web inclusion


```
<li><img src="http://ad.de.doubleclick.net/
ad/N3995.yahoo_DE/B3152413.7;sz=1x1;ord=122
3414745?" width=1 height=1 border=0 style=
'display:none;'>
```

- whole element included from remote service
- images, frames
- advertisement servers, click-through billing services, contents distribution networks

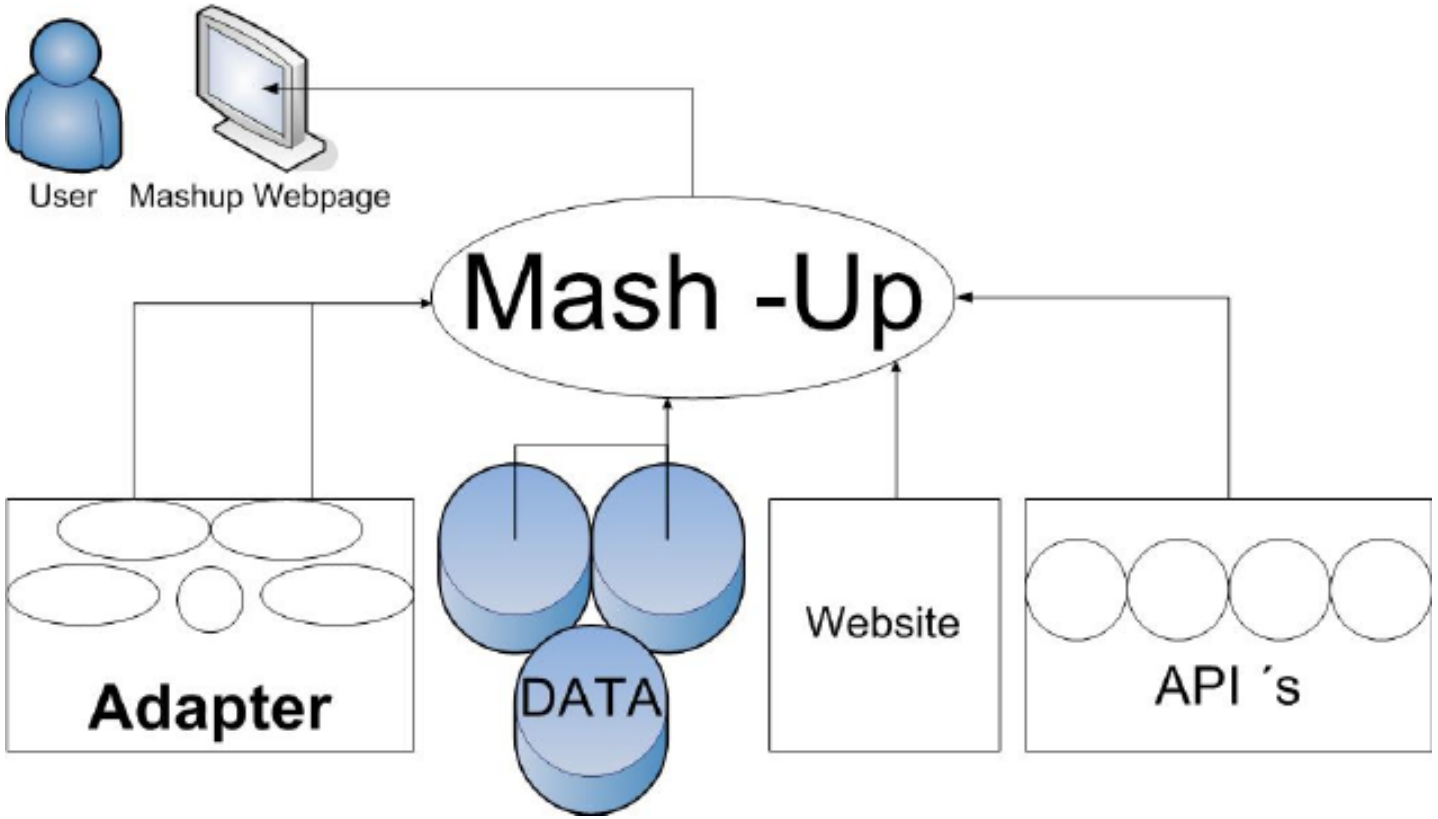
Mashup


```
<script type="text/javascript" src="http://
www.google.com/jsapi?key=ABQIAAAA3nuEoGKhRf
KaTwhFg7OdgxS6mGdPc-RjK_luIMxI5IejX_bbThTmL
SxjVS7PFK_Jwc8dOvCuFMqQOw"></script>
…
google.maps.Event.addListener(markers[46],
"click", function() {…
```

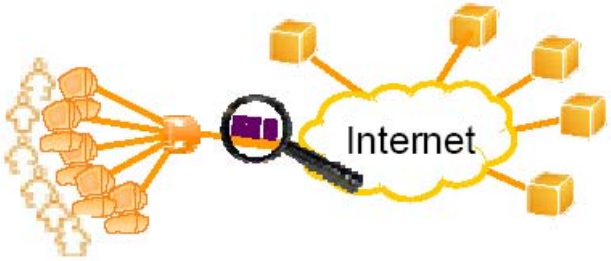
- raw data included from remote services
- local data processing
- maps, images, blogs/information

Mashup Mode of Operation

- Client-side mashup → previous slide
- Server-side mashup → example http://www.mp4-tv.de



Mashup Measurements



- passive measurement
  - observe (aggregate) traffic from real users
  - lots of data, often statistically significant
  - only anonymized traces available / usable due to privacy legislation
  - no correlation back to user actions or Web sites visited
  - no full address visibility required for CIDR prefix investigations

- active measurement
  - injection of IP packets or TCP data transfers
  - measurement of Internet (not Web) characteristics
    - latency, packet loss, re-ordering

- actively initiated measurements
  - defined Web workload
    - list of elements to retrieve
    - list of sites to visit
  - concentrate on service rather than packet level
  - observe latency, download speeds
  - analysis of IP addresses, networks and service structures

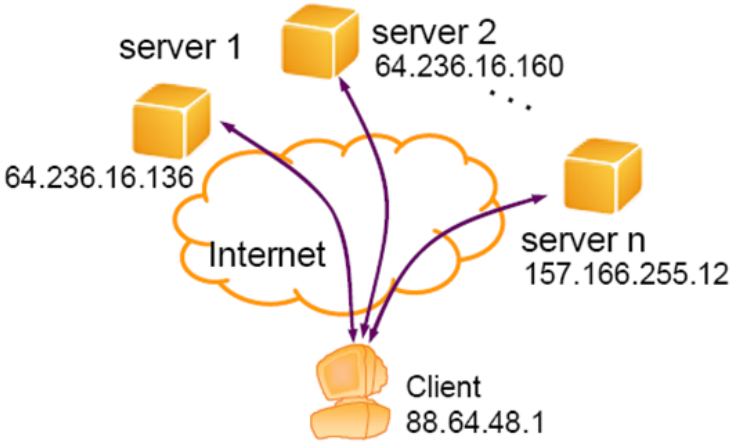
Mashup Measurements

- actively initiated measurements
- visit 100 Web sites most popular in the US according to alexa.com (06/2008)
- visited homepages only
- automated process: for each site do
  - start packet trace
  - open browser to load home page
  - close browser after home page has loaded (or after 1min timeout)
  - stop and store packet trace
- observed measures
  - traffic (rates, volumes, number of packets)
  - locality structure (number of hosts, network prefixes, AS numbers, DNS SLDs)
  - analysis of CDN usage

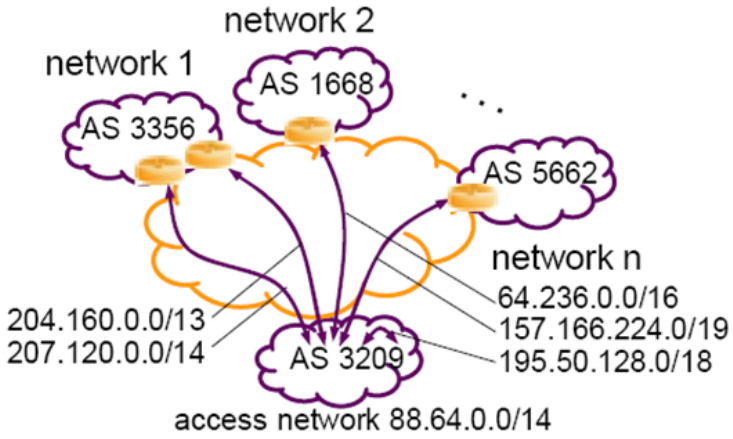
Sites	connections	packets	unique hosts	unique NPs	unique ASs
100	2294	103046	492	234	157

Mashup Measurements

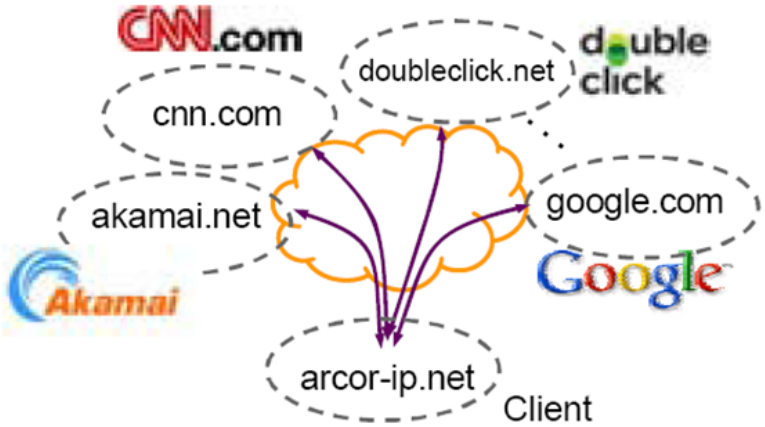
- Locality Notions



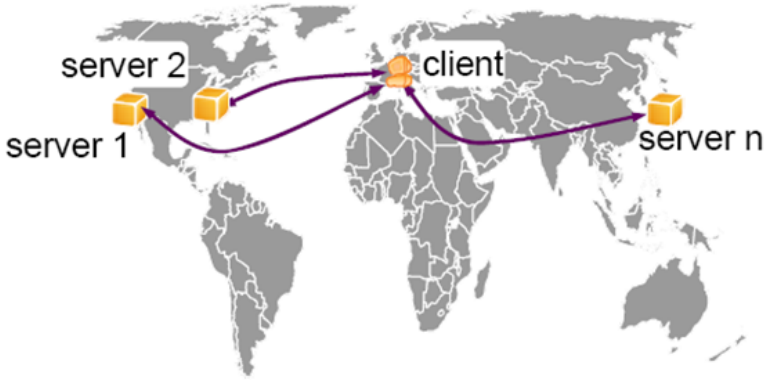
Hosts



Routing Domains (NPs, ASs)



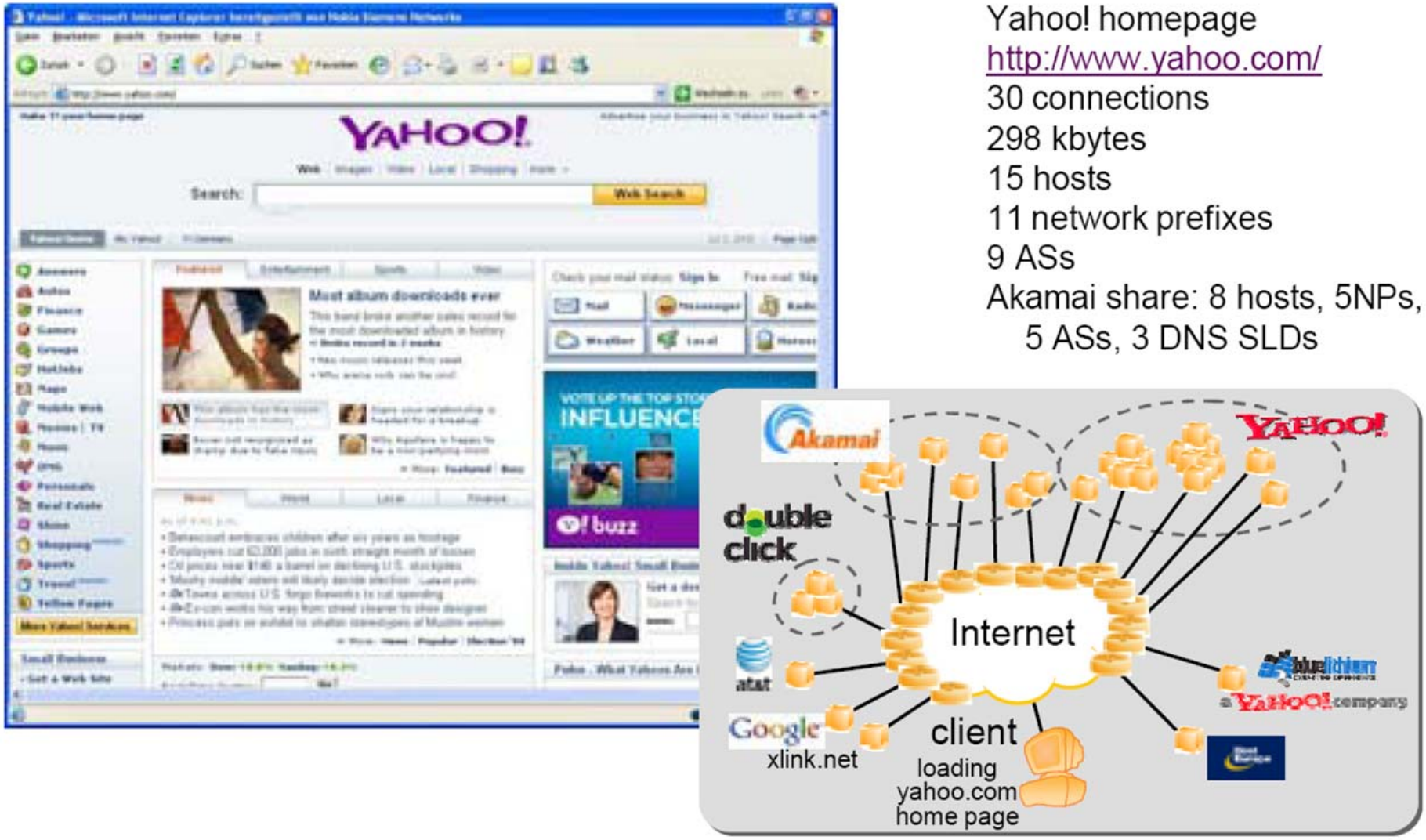
Organizations



Geographic Locations

Mashup Measurements

- Locality Example

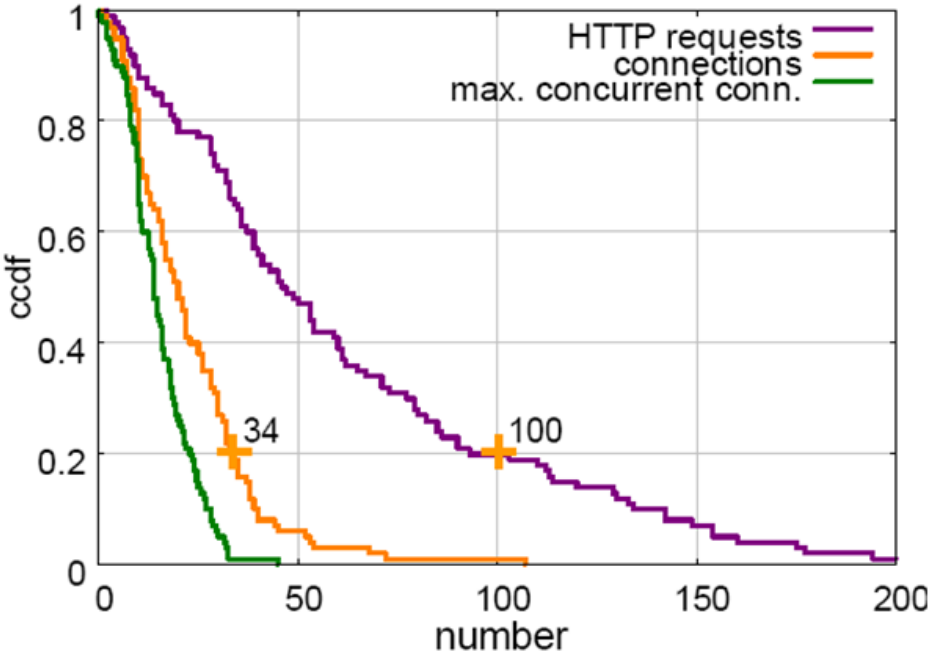


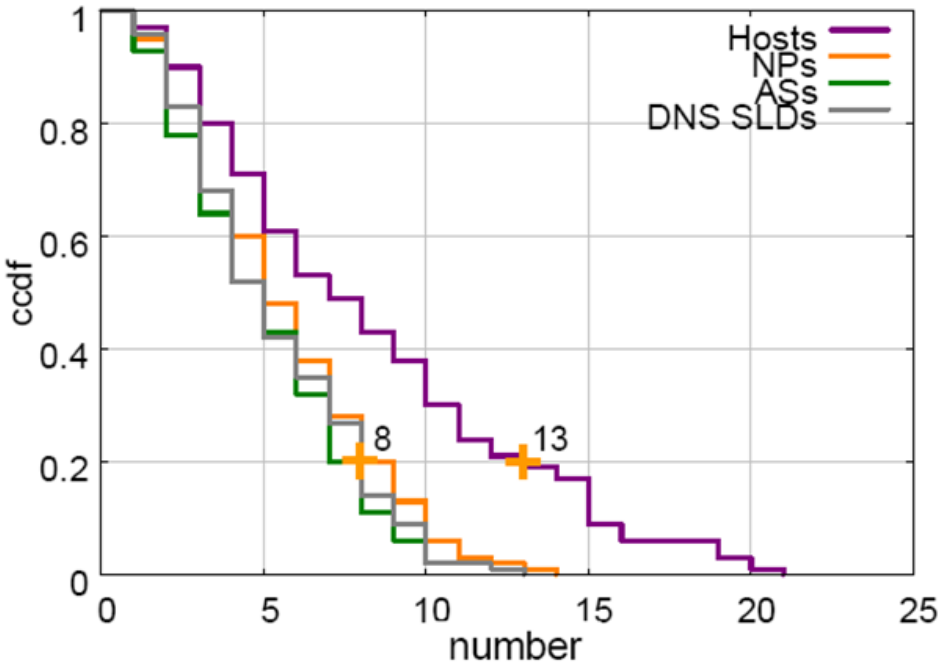
Yahoo! homepage
http://www.yahoo.com/
30 connections
298 kbytes
15 hosts
11 network prefixes
9 ASs
Akamai share: 8 hosts, 5NPs,
5 ASs, 3 DNS SLDs

Mashup Measurements

- Page size and locality measurement results

	Size	Number of Connections		Number of	HTTP	Network	AS	DNS
	(Bytes)	total	max. conc.	Hosts	Requests	Prefixes	Numbers	SLDs
min.	943	1	1	1	2	1	1	1
average	600k	22.94	15.3	8.24	62.2	5.65	5.04	5.15
max.	11.7M	107	45	21	284	14	13	15

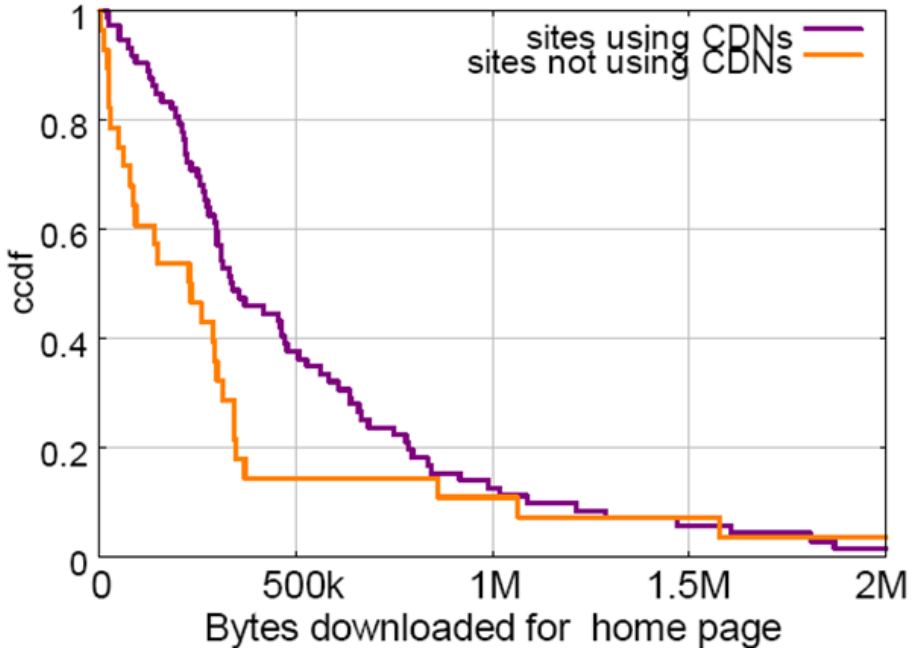


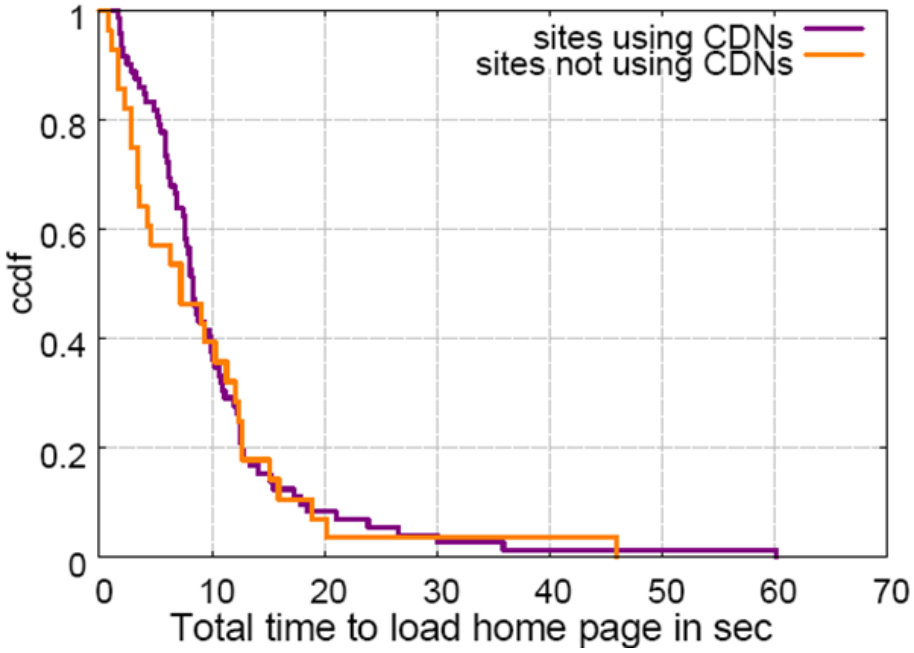


Mashup Measurements

- Web page load speed-up by means of CDNs

	Page Size in Bytes	Number of Hosts	ASs	Data Rate in bit/s	Latency in sec
Average (all)	599k	8.2	5.0	286k	10.0
without CDN	465k	3.9	2.4	196k	9.2
with CDN	651k	9.9	6.1	320k	10.4



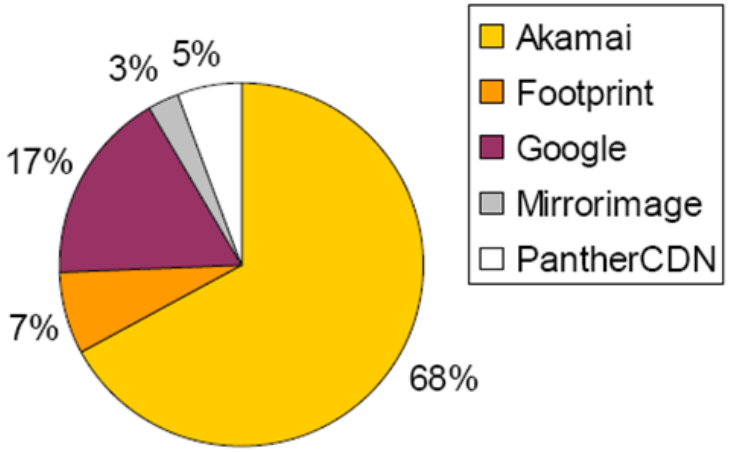


Mashup Measurements

- CDN usage

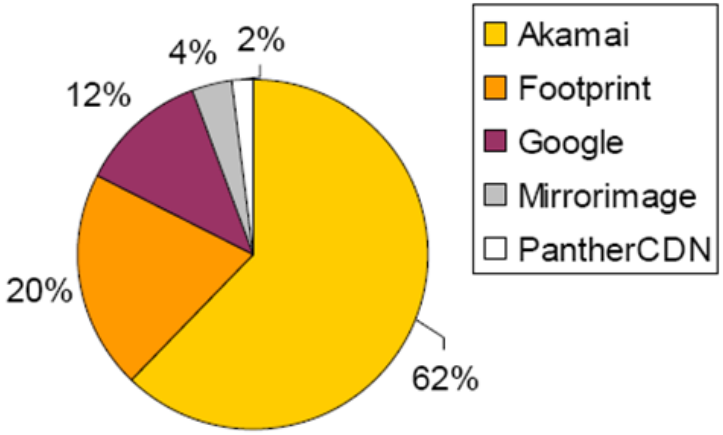
CDN share of connections

CDN share of total: 35%



CDN share of volume

CDN share of total: 46%



CDN domains

Akamai	= akamai.net, akamaiedge.net, akadns.net, akam.net	in Arcor network
Footprint	= footprint.net	via arcor-ip.de, level3.net
Google	= l.google.com	via level3.net
Mirrorimage	= mirrorimage.net, instacontent.net, mirror-image.net	via google.com
Panthercdn	= panthercdn.com	via mii.net, ripe.net
		via ripe.net

Conclusions

- Mashups are quickly building up new services
- Mashups are influencing the Internet usage experience of end users
- Resource distribution in the Internet is reflected in a highly distributed service platform approach
- Networking of client machines requires a large number of concurrent web sessions, which is challenging in NAT environments
- Geographical, organizational and technical distribution of information sources can cause instability, makes the user’s understanding of the service components impossible and leads to traffic matrix explosion
- Service element based charging, QoS optimization of network resources as well as efficient and comprehensive caching strategies are prohibited

Simple Object Access Protocol (SOAP)

Contents - SOAP

History of SOAP

- Times before SOAP
  - RPC
  - COM
  - CORBA
  - …
- Cooperation of Dave Winer (Mr. RSS) and Microsoft
  - Development of XML-RPC
  - Later on Don Box (DevelopMentor) joined the team
  - Simple Object Access Protocol (SOAP) version 0.9
  - 1999 SOAP version 1.0

XML-RPC

- Idea
  - Transfer of the Remote Procedure Call (RPC) via HTTP
  - Encoding of data in XML

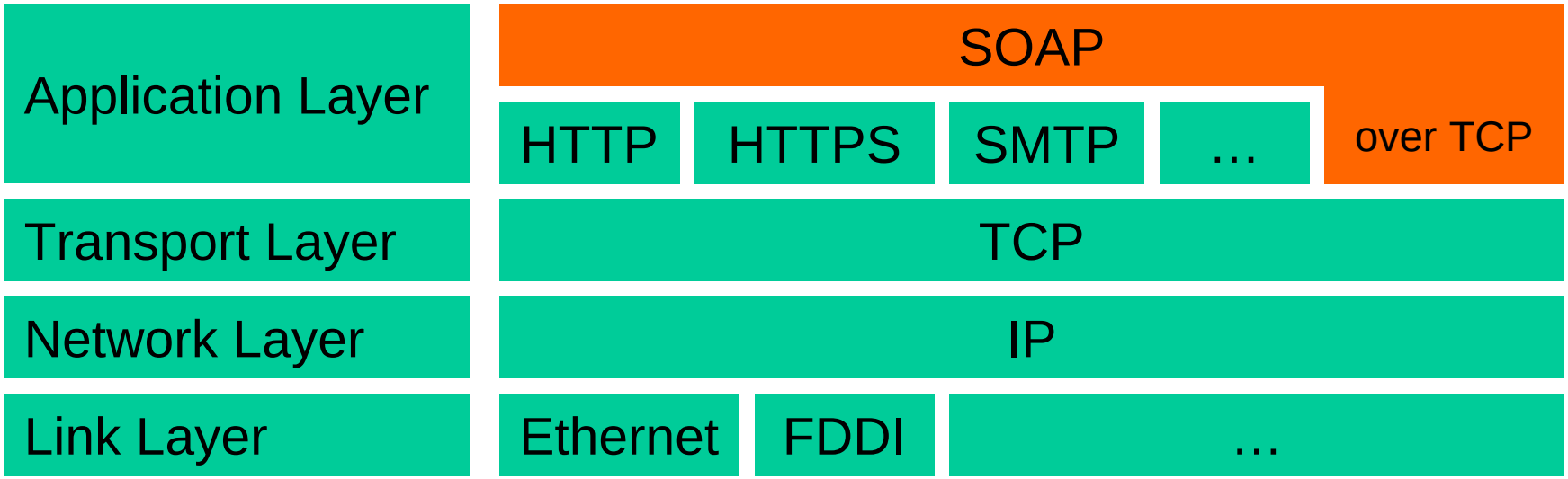
- Example XML-RPC

Request	Response
<?xml version="1.0"?> <methodCall> <methodName>MathService.multiplication</methodName> <params> <param> <value><int>5</int></value> </param> <param> <value><int>6</int></value> </param> </params> </methodCall>	<?xml version="1.0"?> <methodResponse> <params> <param> <value><int>30</int></value> </param> </params> </methodResponse>

Development of SOAP

- Potential of SOAP became aware
  - 2000 IBM joined
  - Submission of specification to the W3C
  - SOAP 1.1 – IBM, Microsoft, DevelopMentor (Don Box) and UserLand Software (Dave Winer)
- W3C Working Group
  - 2002 SOAP 1.2 becomes a W3C recommendation

SOAP in the Internet Protocol Suite



SOAP

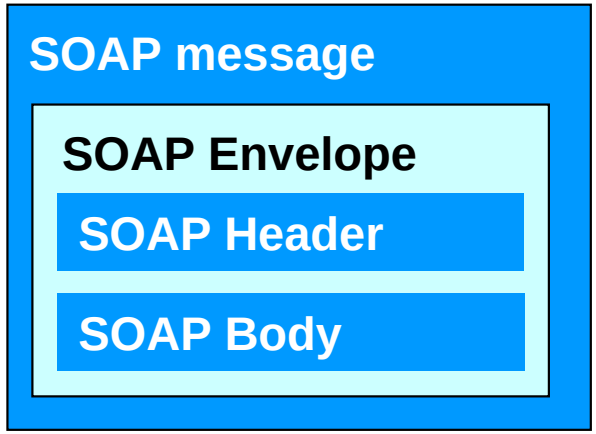
- SOAP offers a simple and light-weight mechanism for the exchange of structured and type-casted information between peers (communication partners) in a decentralized and distributed system

SOAP Message Components

- Components of a SOAP message
  - SOAP Envelope
    - Defines the content of the message, names, namespaces
  - SOAP Encoding Rules
    - Defines the mechanism for serialization of the datatypes
  - SOAP Message Exchange Patterns (MEP)
    - Defines the communication model, e.g. SOAP request-response-MEP
  - RPC representation
    - Defines rules/conventions for representing of remote procedure calls and responses

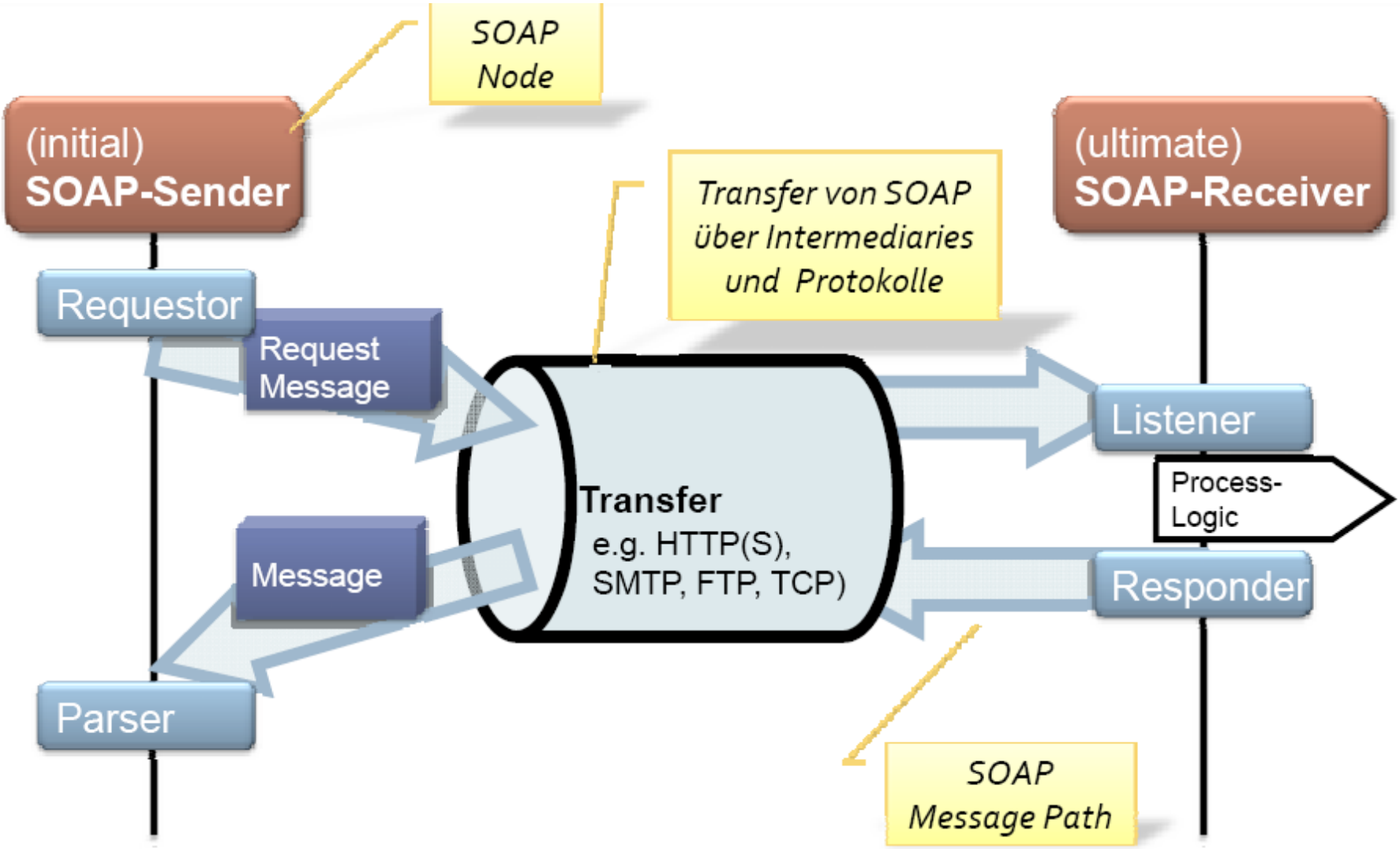
SOAP Message Components - SOAP Envelope

- SOAP Envelope
  - Header (optional)
    - For adding data which do not belong to the payload
    - For transmitting control information (e.g. routing, security, …)
  - Body
    - Application data / payload which is transferred between a SOAP sender and a SOPA receiver (end-to-end principle)
  - Defines namespaces




```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Header>
   </soap:Header>
   <soap:Body>
   </soap:Body>
</soap:Envelope>
```

SOAP Scenario



SOAP Examples

- SOAP-Request via HTTP-POST

```
POST /ScientificCalculator HTTP/1.1
Host: www.mathcalculators.org
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn

<env:Envelope env:soap=" xmlns:env="http://www.w3.org/2003/05/soap-envelope">
      <env:Body>
            <m:Multiplication xmlns:m="http://www.mathcalculators.org/sc_rev3">
                  <m:int17>5</m:int17>
                  <m:int17>6</m:int17>
            </m:Multiplication>
      </env:Body>
</env:Envelope>
```

SOAP Examples

- SOAP-Response via HTTP

```
HTTP/1.1 200 OK
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn

<env:Envelope env:soap=" xmlns:env="http://www.w3.org/2003/05/soap-envelope">
      <env:Body>
            <m:MultiplicationResult xmlns:m="http://www.mathcalculators.org/sc_rev3">
                  <m:int25>30</m:int257>
            </m:MultiplicationResult>
      </env:Body>
</env:Envelope>
```

SOAP Examples

- SOAP-Fault via HTTP

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<soap:Envelope xmlns:soap=" xmlns:env="http://www.w3.org/2003/05/soap-envelope ">
      <soap:Body>
            <soap:Fault>
                  <faultcode>SOAP: MustUnderstand</faultcode>
                  <faultstring>SOAP Must Understand Error</faultstring>
            </soap:Fault>
      </soap:Body>
</soap:Envelope>
```