Le protocole HTTP

Didier DONSEZ

Université Joseph Fourier (Grenoble 1)

PolyTech Grenoble – LIG/ERODS

Didier.Donsez@imag.fr

Didier.Donsez@ieee.org

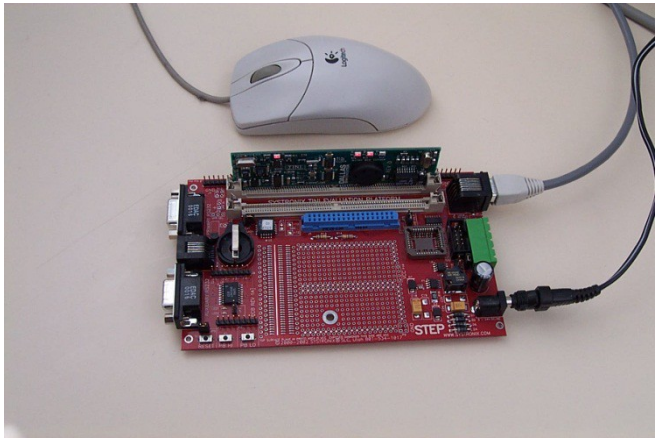
Le Protocole HTTP

- HTTP : HyperText Tranfert Protocol *(RFC 1945 et 2068)*
  - protocole de rapatriement des documents
  - protocole de soumission de formulaires
- Fonctionnement *(très simple en HTTP/1.0)*
  - Connexion TCP/IP
  - Demande (GET) d ’un document
  - Renvoi du document (status=200) ou d ’une erreur
  - Déconnexion TCP/IP
- *Cependant*
  - *dialogue plus complexe en cas d ’identification*
  - *optimisation :*
    une série de plusieurs requêtes sur une connexion
    - Connexion « KeepAlive » de HTTP/1.1 *(RFC 2068)*

Un protocole omniscient

- IT Servers

- Embedded Servers
  - Web of Things



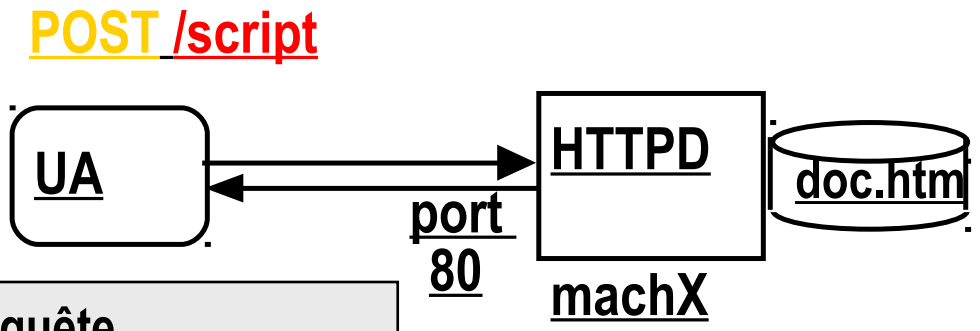
Dialogue HTTP

- Protocole en mode de lignes de caractère
  - le plus basic de butineur (User Agent) HTTP :
    - telnet www.google.com 80
- Types de dialogue
  - Récupération d ’un document
    - méthode GET
  - Soumission d ’un formulaire
    - méthodes GET ou POST
  - Envoi de Document et Gestion de Site
    - méthodes PUT, DELETE, LINK, UNLINK
  - *Gestion de proxy/cache et REST*
    - *méthode HEAD (récupération des informations sur le document)*

Dialogue HTTP

exemple de requête HTTP (méthode POST)

Didier Donsez, 1993-2014, HTTP



Entête de la requête

L'UA envoie la Requête

```
POST /script HTTP/1.0        méthode,chemin,version
Accept: www/source           documents acceptés
Accept: text/html
Accept: image/gif
Accept: image/jpeg
User-Agent:  Lynx/2.2  libwww/2.14
* une ligne blanche *
name1=value1&
name2=value2
```

Corps de la requête

le Serveur retourne la Réponse

```
HTTP/1.0 200 OK              ligne de status
Date: Wed, 02Feb97 23:04:12 GMT
Server: NCSA/1.1
MIME-version: 1.0
Last-modified: Mon,15Nov96 23:33:16 GMT
Content-type: text/html      type du document retourné
Content-length: 2345         sa taille
    * une ligne blanche *
<HTML><HEAD><TITLE> ...
```

Entête de la réponse

Corps de la réponse

Format de la requête

- Envoyé par l'UA au serveur

  <Méthode> <URI> HTTP/<Version>

  [<Champ d 'entête>: <Valeur>]

  [<tab><Suite Valeur si >1024>]

  ligne blanche

  [corps de la requête pour la méthode POST]

```
GET /docu2.html HTTP/1.0
Accept: www/source
Accept: text/html
Accept: image/gif
User-Agent:  Lynx/2.2  libwww/2.14
 * une ligne blanche *
```

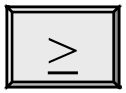
```
POST /script HTTP/1.0
Accept: www/source
Accept: text/html
Accept: image/gif
User-Agent:  Lynx/2.2  libwww/2.14
Content-Length: 24
   * une ligne blanche *
name1=value1&
name2=value2
```

Méthodes de la requête (i)

- GET
  - demande pour obtenir des informations et une zone de données concernant l ’URI
- HEAD
  - demande pour seulement obtenir des informations concernant l ’URI *(plutôt pour)*
- POST
  - envoie de données (contenu du formulaire vers le serveur, requête SOAP …) situées dans le corps.
- PUT
  - enregistrement du corps de la requête à l ’URI indiqué
- DELETE
  - suppression des données désignées par l ’URI

Méthodes de la requête (ii)

- OPTIONS
  - demande des options de communication disponibles
- TRACE
  - retourne le corps de la requête intacte (déboguage)
- CONNECT



- + Nouvelles extensions de WebDAV
  - PROPFIND, PROPPATCH, MKCOL,COPY, MOVE, LOCK, UNLOCK
- + Nouvelles extensions de HTTP/U et HTTP/MU
  - NOTIFY, … (UPnP), NOTIFY (CoAP)

Format de la réponse

- Réponse envoyé par le serveur à l'UA

  HTTP/<Version> <Status> <Commentaire Status>

  [< Champ d 'entête >: <Valeur>]

  [<tab><Suite Valeur si >1024>]

  Ligne blanche

  Document

```
HTTP/1.0 200 OK
Date: Wed, 02Feb97 23:04:12 GMT
Server: NCSA/1.1
MIME-version: 1.0
Last-modified: Mon,15Nov96 23:33:16 GMT
Content-type: text/html
Content-length: 2345
     * une ligne blanche *
<HTML><HEAD><TITLE> …
</BODY></HTML>
```

Status des réponses HTTP *(RFC2068)*

- Code de réponse donné par le serveur au UA
  - 100-199 Informationnel
    - 100 : Continue (l'UA peut envoyer la suite de la requête), ...
  - 200-299 Succès de la requête UA
    - 200: OK, 201: Created, 204 : No Content, ...
  - 300-399 Redirection de la Requête UA
    - 301: Redirection, 302: Found, 304: Not Modified, 305 : Use Proxy, ...
  - 400-499 Requête UA incomplète
    - 400: Bad Request, 401: Unauthorized, 403: Forbidden, 404: Not Found
  - 500-599 Erreur Serveur
    - 500: Server Error, 501: Not Implemented,
    - 502: Bad Gateway, 503: Out Of Resources (Service Unavailable)

Champs d'Entêtes HTTP

- Types de champs d ’entête
  - Général, UA, Serveur, Entité
- Exemple
  - Content-Type = type MIME du l'entité renvoyé
  - Content-Length = taille du l'entité en octet
  - Cache-Control = contrôle du caching.
  - Last-Modified = date de dernière modification de l'entité.
  - Accept = type MIME visualisable par l ’agent
  - Accept-Language : liste de langues (fr, en, …)
  - Cookie = cookie stocké et retourné par l'UA
  - Set-Cookie = créer ou modifie un cookie sur l'UE
  - Range = zone de l'entité à renvoyer

Format d 'encodage : **x-www-form-urlencoded**

- Codage des « paramètres » : form HTML, boutons HTML, Imagemap, …
  - GET : dans l'URL
  - POST : dans le corps de la requête
  -

- Caractères « standards » 7 bits
- Espace → **+**
- Tous les caractères spéciaux et accentués → **%code ascii**
  - **@ %40**
  - **é %e9**
- Les entrées des formulaires sont encodés dans une chaîne composée de paires (nom de l 'entrée)=(valeur de l 'entrée) séparé par de &
  - **nom=Dupont+Jean&adresse=3+rue+de+la+Gait%e9%0a75014+Paris**

Suivi de Sessions avec HTTP
(Session Tracking)

- Motivations :
  - La notion de session est importante dans une application conversationnelle
    - commerce électronique
      - « j 'ajoute ce produit à mon panier (existant)»
  - Cependant HTTP est un protocole « stateless »
    - le serveur ne maintient pas d 'informations liées aux requêtes précédentes d 'un même client.
    - HTTP est donc « sessionless »
  - Comment implanter la notion de session sur plusieurs requêtes HTTP
    - documents, CGI, SSS, Servlet/JSP, ASP, PHP, …

Suivi de Sessions avec HTTP
(Session Tracking)

- Méthodes
  - Le serveur génère un identificateur de session et associe un état (et une date limite de validité) à une session
  - Le client renvoie l ’identificateur de session à chaque requête HTTP vers le serveur

- Echange et Stockage de l’identificateur de session
  - *Input HIDDEN dans les formulaires*
  - Réécriture des URLs (EXTRA_PATH)
  - Cookies (déactivable)
  - *Identificateur de session SSL (Secure Socket Layer)*

Suivi de Session
la ré-écriture des URLs

- L ’identifiant de session est encodé dans les URLs des documents HTML retourné par le serveur.

  http://localhost:8080/servlet/cart?PROD_ID=383

  devient

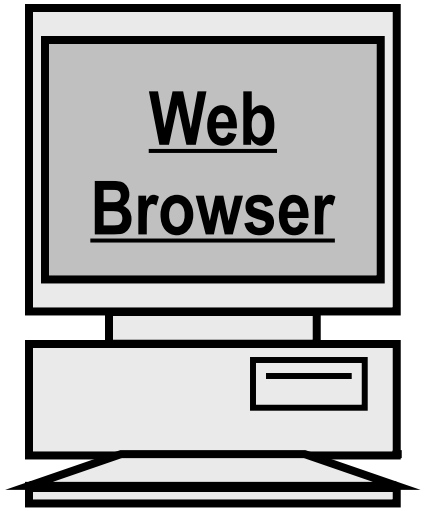
  http://localhost:8080/servlet/cart;jsessionid=To1128mC33718557521577075At?PROD_ID=383

- Codage différent d’un serveur à l’autre.
  - **field_sessionID = "JServSessionIdroot";** **//Apache/Jserv**
  - **field_sessionID = ";jsessionid=";** **//Jakarta-Tomcat**
  - **field_sessionID = "JIGSAW-SESSION-ID";** **//Jigsaw web servers**
  - **field_sessionID = ";$sessionid$";** **// WebSphere**
- Limites
  - URL générée par un script (=>programmation)
  - ou parsing des documents HTML retournés
    - mais disfonctionnement en présence de scripts JavaScript ou VBScript générant eux aussi des URL !

Suivi de Session
les Cookies [RFC2109]

- chaîne décrivant l'état d'une session
  - NAME=VALUE;
  - expires=DATE;
  - path=PATH_HEAD; / << /foo << /foobar ou /foo/bar.html
  - domain=DOMAIN_TAIL; fr << mycomp.fr << sales.mycomp.fr
- stocké sur le client
  - Limite
    - 300 cookies simultanées par client, 20 cookies par serveur ou domaine, 4Ko par cookie (limite la taille des VALUEs)
- communiqué dans les entêtes de requêtes (Cookie:) et dans les entêtes des réponses HTTP (Set-Cookie:)
- accessible par les scripts JavaScript dans une page HTML

Positionnement des Cookies (i)



Web browser' cookies table

VisitorID=12345; path=/;
SID_book=88; path=/book;
SID_music=999; path=/music;

GET /

Set-Cookie: VisitorID=12345; path=/;

GET /book
Cookie: VisitorID=12345;

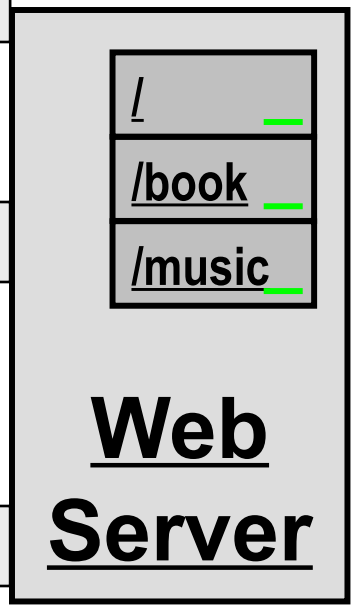
Set-Cookie: SID_book=88; path=/book;

GET /book/order
Cookie: VisitorID=12345;
Cookie:SID_book=88;

GET /music
Cookie: VisitorID=12345;

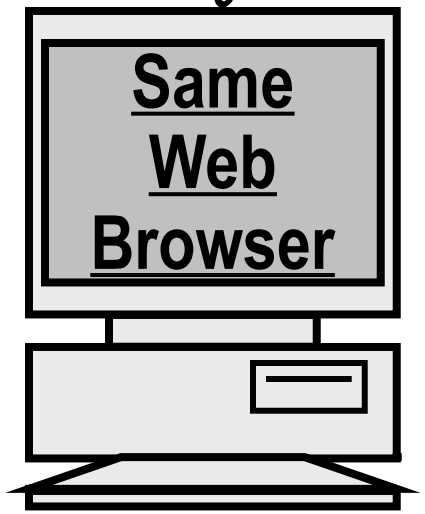
Set-Cookie: SID_music=999; path=/music;

GET /music/buy
Cookie: VisitorID=12345;
Cookie: SID_music=999;



Positionnement des Cookies (ii)

Un autre jour !



Web browser' cookies table

VisitorID=12345; path=/;

GET /
Cookie: VisitorID=12345;

GET /book
Cookie: VisitorID=12345;

Set-Cookie: SID_book=101; path=/book;

GET /book/logout
Cookie: VisitorID=12345;
Cookie:SID_book=101;

Set-Cookie: SID_book=0; path=/book; \
expires=0

GET /
Cookie: VisitorID=12345;

/
/book
/music

Web Server

HTTPS *(S comme Secure)*

- Motivation
  - Sécuriser (authentification, confidentialité) l'accès à un service Web
    - Clé privée + Certificat X509 du serveur (obligatoire)
    - Clé privée + Certificat X509 de l'usager (optionnel)
- SSL/TLS
  - Phase 1 : Authentification du serveur et/ou de l'usager par PKI
  - Phase 2: Chiffrage avec une clé (secrète) symétrique de session
  - Phase 2bis : Reprise après déconnexion
- HTTP over SSL (TLS)
  - URL : https://host/document (Port TCP par défaut : 443)
- HTTPS dans le JDK
  - JSSE (javax.net.ssl.*)
  - Classe javax.net.HttpsURLConnection
  - JCE (Classe Keystore, ...)

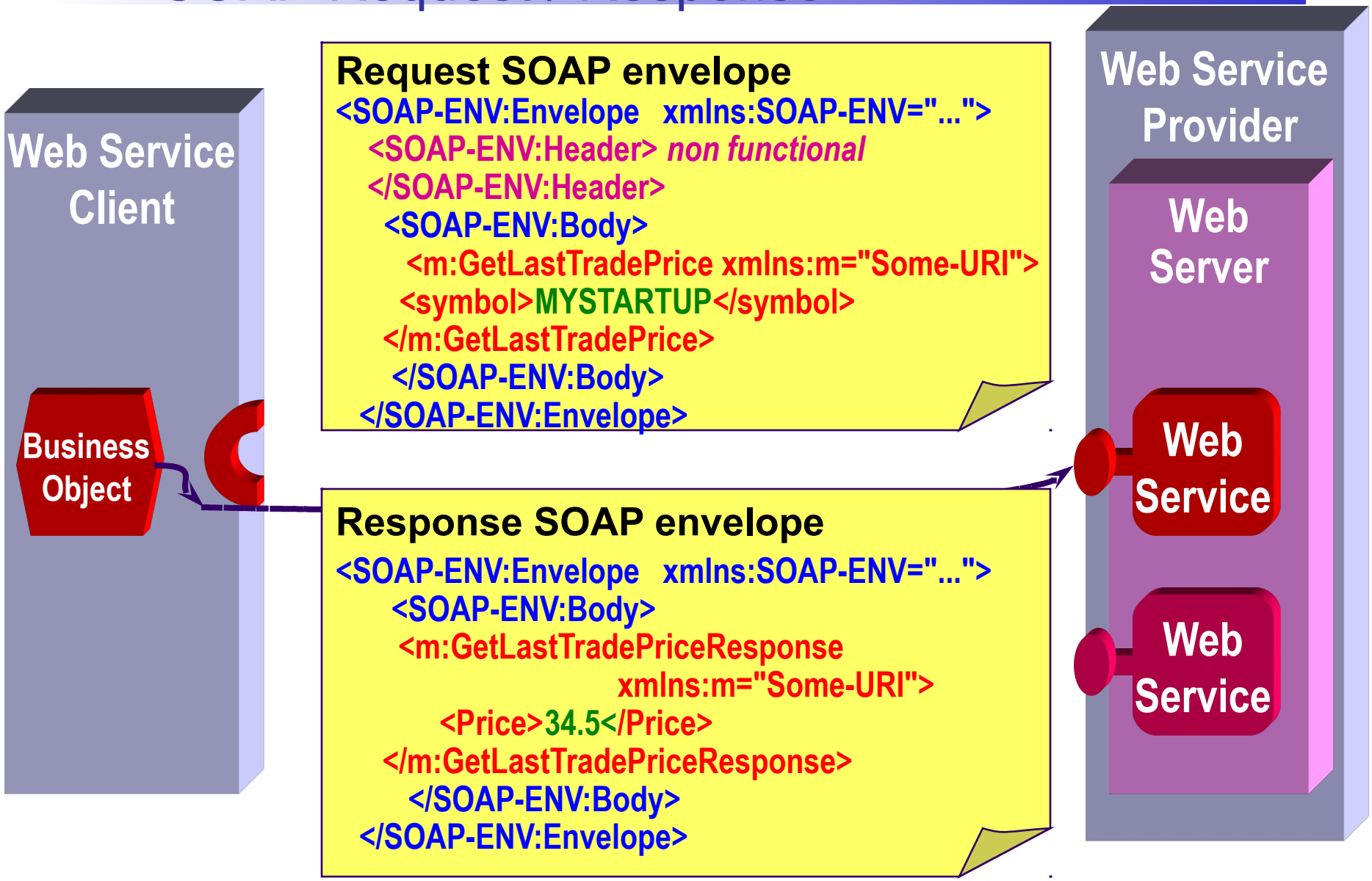
XML-RPC vs SOAP vs REST

- Motivations
  - Program-to-Program
  - B2B, AJAX, …
- XML-RPC
  - HTTP POST
  - Simple encoding schema
- SOAP
  - operation-centric
  - WSDL
- REST
  - more URL-centric

Web Services

- Motivations
  - RPC sur les protocoles IETF (HTTP, SMTP, …)
  - Pour l'EAI, B2B, BPM (BPEL), ...
- Principes
  - WSDL : Description public en XML Schema
    - IDL + Entry Points
  - SOAP : Encodage (Mashalling) en XML
    - Opération, Paramêtres, Résultats, Erreurs/Fautes
    - + Entête non-fonctionnel (sécurité, session, transaction, …)
  - Liaison (*binding*) avec un protocole (HTTP POST, …)
    - Entête et corps des messages
      - Par exemple : code d'erreur HTTP 404

SOAP Request / Response



Exemple de requête SOAP sut HTTP

Demande de cotation à un serveur

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>MYSTARTUP</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemple de réponse SOAP sur HTTP

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

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
           <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

REST = REpresentational State Transfert

- Style d'architecture de type client-serveur
  - 3 Levels of Richarson Maturity Model
- Considère des collections de ressources (maintenant leur état)
  - Adressables pour une URI
    - `http://www.boeing.com/aircraft`
      `http://www.boeing.com/aircraft/747`
  - Accessible via les méthodes (verb) HTTP (CRUD)
    - PUT, GET, POST, DELETE, HEAD, OPTION, PATCH
    - Status code (200, 201, 409, ...), Header (Accept, Location, ...)
  - Représentation formateé en fonction du client
    - Formats : XML, HTML (fragment), JSON, JPEG,…
- Outils
  - Caches, Proxies, ...

Exemple

Classiquement

- GET http://example.com/book?action=showall
- GET http://example.com/book?action=show&isbn=0596529260
- GET http://example.com/book?action=modify&isbn=0596529260&price=49.99
- POST http://example.com/book?action=modify
- GET http://example.com/book?action=delete&isbn=0596529260

Avec REST

- GET http://example.com/book
- GET http://example.com/book/0596529260
- POST http://example.com/book/0596529260
- PUT http://example.com/book/0596519260
- DELETE http://example.com/book/0596519260

Exercice : comment modéliser un panier d'un client ?

Solution : POST http://example.com/cart

Modèle de Maturité de Richardson

A lire http://martinfowler.com/articles/richardsonMaturityModel.html

Level 0

Communication client / serveur via le protocole HTTP

1 URL / 1 type de verbe HTTP (en général POST)

Level 1: Resources

Plusieurs URLs hiérarchisées

- Représente des collections

toujours 1 seul verbe HTTP (en général POST)

Level 2: HTTP Verbs

Verbes HTTP (GET, POST, PUT, DELETE, OPTION, HEAD, PATCH)

Status code HTTP

Level 3: Hypermedia Controls

HATEOAS (Hypertext As The Engine Of Application State)

CRUD et Verbes REST (RMM Level 2)

- Create
  - PUT if and only if you are sending the full content of the specified resource (URL).
  - POST if you are sending a command to the server to create a subordinate of the specified resource, using some server-side algorithm.
- Retrieve = GET, HEAD.
- Update
  - PUT if and only if you are updating the full content of the specified resource.
  - POST if you are requesting the server to update one or more subordinates of the specified resource.
  - PATCH for semantic changes
- Delete = DELETE.
- Info = OPTION
  - Used to request information about the communication options of the resource you are interested in. It allows the client to determine the capabilities of a server and a resource without triggering any resource action or retrieval.

http://jcalcote.wordpress.com/2008/10/16/put-or-post-the-rest-of-the-story/

HATEOAS

Hypertext As The Engine Of Application State

API de navigation dans les collections et les ressources

- *Martin Fowley, RMM Level 3 introduces discoverability, providing a way of making a protocol more self-documenting.*

GET /account/12345 HTTP/1.1

HTTP/1.1 200 OK

```
<?xml version="1.0"?>
<account>
 <account_number>12345</account_number>
 <balance currency="usd">100.00</balance>
 <link rel="self" href="/account/12345" />
 <link rel="deposit" href="/account/12345/deposit" />
 <link rel="withdraw" href="/account/12345/withdraw" />
 <link rel="transfer" href="/account/12345/transfer" />
 <link rel="close" href="/account/12345/close" />
 <link rel="comment" type="application/x.atom+xml" title="Blog comments"
href="/account/12345/comment"/>
 </account>
```

REST Asynchronous Requests (i)¶

Opérations différée ou de longues durées sur des ressources/collections REST

POST /blog HTTP/1.1

<blog><title>Bla Bla<title><author>Me</author><text>Bla Bla Bla Bla !</text></blog>

HTTP/1.1 202 Accepted

Location: /queue/12345

GET /queue/12345 HTTP/1.1

HTTP/1.1 200 OK

<queue>

<status>Pending</status>

<eta>10 minutes</eta>

<link rel="cancel" method="delete" href="/queue/12345"/>

</queue>

...

verbe DELETE
pour annuler l'opération

http://restcookbook.com/Resources/asynchroneous-operations/
https://www.adayinthelifeof.nl/2011/06/02/asynchronous-operations-in-rest/

REST Asynchronous Requests (ii)¶

…

GET /queue/12345 HTTP/1.1

HTTP/1.1 200 OK

<queue>

<status>In progress</status>

<eta>3 minutes, 25 seconds</eta>

</queue>

Plus de verbe DELETE pour annuler l'opération

GET /queue/12345 HTTP/1.1

HTTP/1.1 303 See Other

Location: /blog/20140101-Bla%20Bla

http://restcookbook.com/Resources/asynchroneous-operations/
https://www.adayinthelifeof.nl/2011/06/02/asynchronous-operations-in-rest/

REST PATCH verb

PUT versus PATCH

PUT fournit la nouvelle valeur intégrale de l'entité

PATCH donne des instructions à appliquerr

- JSON Patch http://tools.ietf.org/html/rfc6902

```
PATCH /users/123
Content-Type: application/json-patch
[
   { "op": "replace", "path": "/email", "value": "new.email@example.org" }
]
```

```
PATCH /agenda/123
Content-Type: application/json-patch
[
   { "op": "test", "path": "/a/b/c", "value": "foo" },
   { "op": "remove", "path": "/a/b/c" },
   { "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ] },
   { "op": "replace", "path": "/a/b/c", "value": 42 },
   { "op": "move", "from": "/a/b/c", "path": "/a/b/d" },
   { "op": "copy", "from": "/a/b/d", "path": "/a/b/e" }
]
```

207 Multi-Status

http://williamdurand.fr/2014/02/14/please-do-not-patch-like-an-idiot/

REST Misc

Range

- GET /api/collection
- Range: resources=100-199
-
- OPTIONS /api/collection HTTP/1.1
- HTTP/1.1 200 OK
- Accept-Ranges: resources

Notification (long poll)

- Example : return at least one resource with an ID > 100.

- GET /api/collection
- Range: 100-
- Expect: nonempty-response

REST Worst Practices http://jacobian.org/writing/rest-worst-practices/

REST Misc

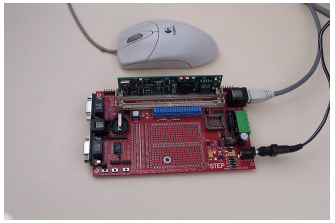
Web Application Description Language (WADL)

```
<application xmlns="http://wadl.dev.java.net/2009/02">
<doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 2.0"/>
<resources base="http://www.apress.com/">
<resource path="{id}">
<param name="id" style="template" type="xs:long"/>
<method name="GET">
<response>
<representation element="book" mediaType="application/xml"/>
<representation element="book" mediaType="application/json"/>
</response>
</method>
<method name="DELETE"/>
</resource>
<resource path="book">
<method name="GET">
<response>
<representation element="book" mediaType="application/xml"/>
<representation element="book" mediaType="application/json"/>
</response>
</method>
</resource>
</resources>
</application>
```

Autre exemple
REST et Web des Objets

Embedded server (low cost, low power) + sensors, RFID readers, …

→ Polling and Pushing with REST



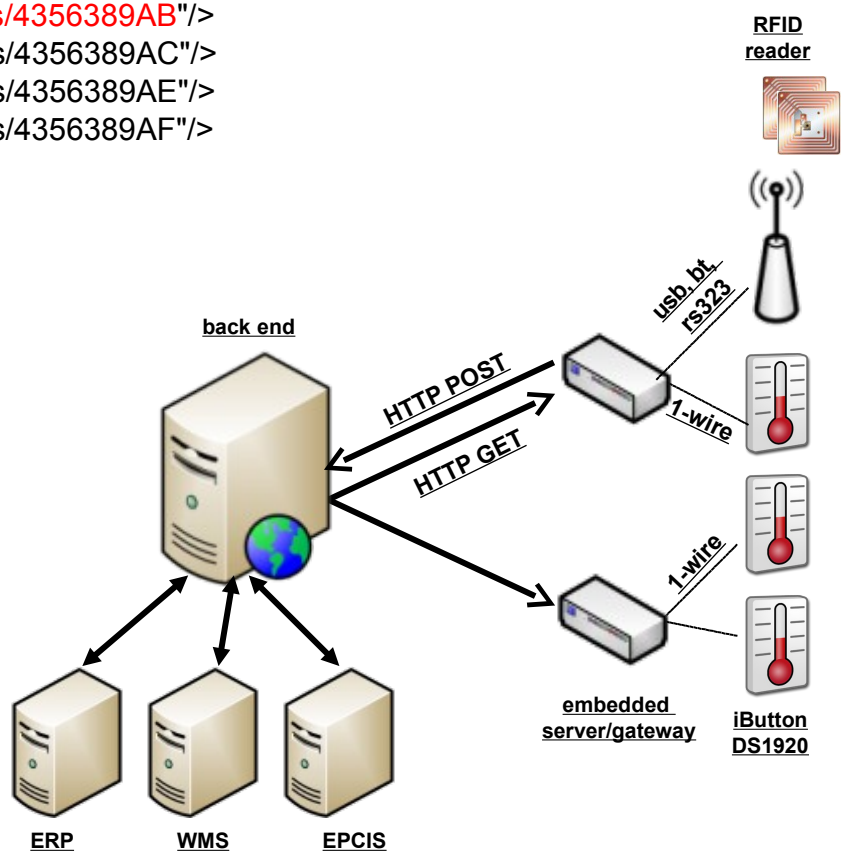
GET /sensors

```
<?xml version="1.0"?>
<s:sensors xmlns:s="http://iotgw-1234.com/sensors"
        xmlns:xlink="http://www.w3.org/1999/xlink">
      <sensor id="00345" xlink:href="http://iotgw-1234.com/sensors/4356389AB"/>
      <sensor id="00346" xlink:href="http://iotgw-1234.com/sensors/4356389AC"/>
      <sensor id="00347" xlink:href="http://iotgw-1234.com/sensors/4356389AE"/>
      <sensor id="00348" xlink:href="http://iotgw-1234.com/sensors/4356389AF"/>
</s:sensors>
```

GET /sensors/4356389AB

```
<?xml version="1.0"?>
<s:sensor xmlns:p="="http://iotgw-1234.com/sensors"
        xmlns:xlink="http://www.w3.org/1999/xlink">
        <id>4356389AB</id>
        <type>DS1920</type>
        <value>344</value>
        <error>0.5</error>
        <unit>K</unit>
        <timestamp> ... <timestamp>
</s:sensor>
```

Format: EEML, KML, … voir Pachube



Cache Web

- Motivation
  - Usager
    - Améliorer les performances de l'UA en utilisant les entités présents dans le cache local
  - ISP et CDN
    - Soulager le réseau opérateur en cachant les entités demandés par les usagers du sous-réseau
  - Services
    - Soulager les serveurs applicatifs
- Entités à cacher
  - Textes (HTML, XML, JSON…) statiques ou générés
  - Contenus multimédia (complet/fragment)
- Champs d'entête
  - `Cache-Control, Expires, Last-Modified, If-Modified-Since, ...`
- Outils: Squid, Memcached, ...

Content Delivery Network (CDN)

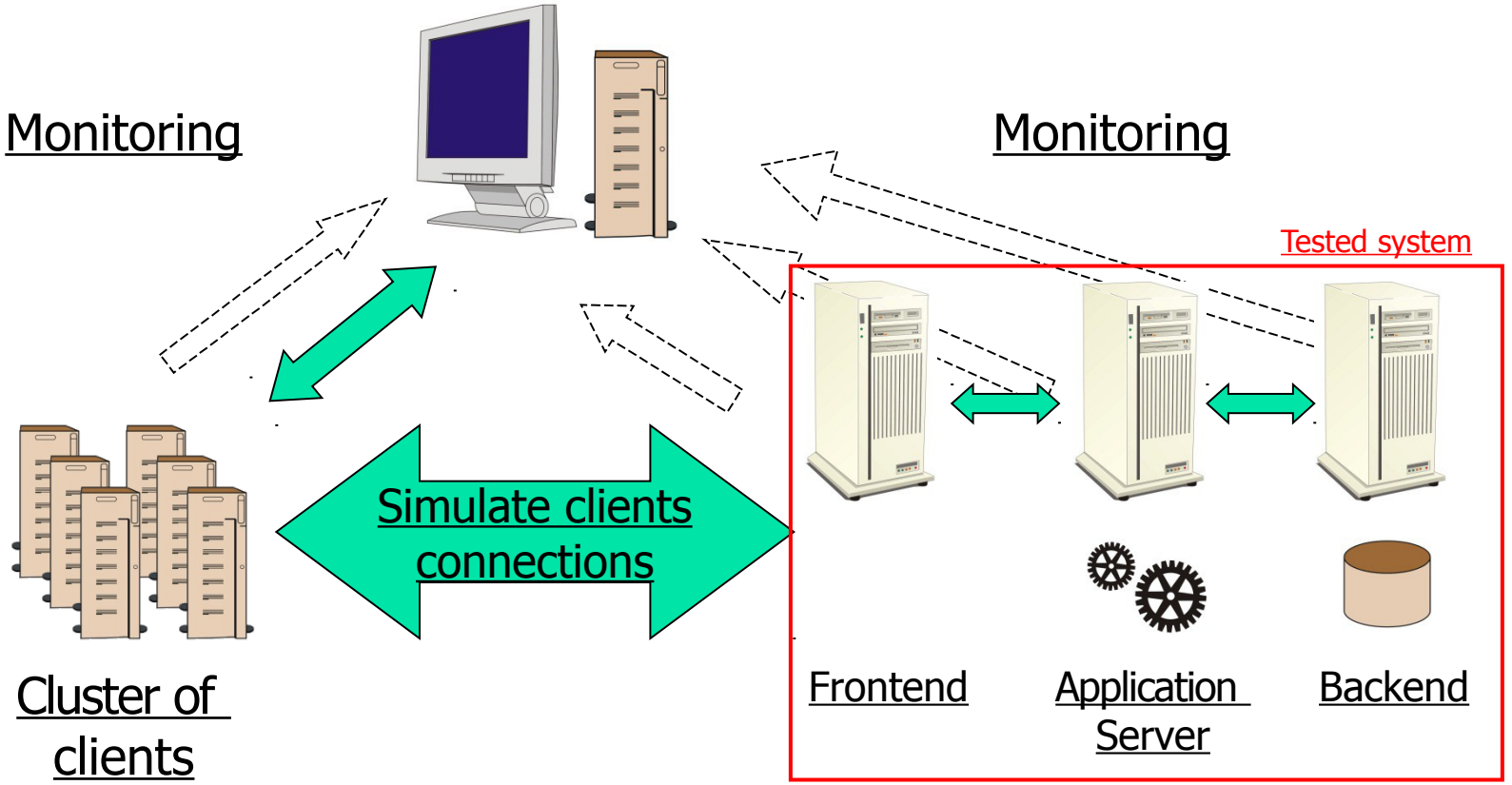
- Web cache aux points de présence du(des) FAI
  entre le fournisseur et ses clients
- Documents cachés
  - Statique, Stream, Fragments de compositions dynamiques (ESI)
- Avantages
  - Latence réduite, décharge du serveur fournisseur (allocation de serveurs à la demande, …), décharge du réseau central FAIs
- Inconvénients
  - Usage de DNS dynamique dissociée pour cachable et non cachable
  - Réécriture dynamique d'URL
- Acteurs
  - Akamaï (+Digital Island) ~80% du marché
    - 2010 : 73 000 globally distributed servers, handles 15 to 30 percent of all Web traffic
  - Adero, Mirror Image, Inktomi, AWS, Azure …

Test de performance / charge

- Motivation
  - Vérification du niveau de charge support avant la mise en production
  - Dimensionnement de l'infrastructure à acheter/louer
- Benchmarks
  - TPC-W (Bookstore online), SPECWeb, Rubis (eBay auction) , Rubos (PHP News SlashDot), CSIRO (Stock online), Benchlab (Web Mobile)
- Analyse et diagnostique de la charge
  - Détecter quel composant (HTTPD, EJB, MT, SGBD, Mailhost, MQ …) en créant l'embouteillage (*bottleneck*)
    - *Mauro Andreolini, Valeria Cardellini, Michele Colajanni: Benchmarking Models and Tools for Distributed Web-Server Systems. Performance 2002: 208-235*
- Outils
  - Apache Benchmark Tool (ab), Siege, Apache JMeter, IBM SiteLoad, LoadRunner, TestMaker, Grinder, Gatlin, CLIF

Test de performance / charge (ii)

- Injecteur de charge
  - 1à N clients en parallèle qui émulent M WebSurfers sur le site
    - Script définissant la navigation (automate, temporisation, …)



Répartition de Charge *(Load Balancing)* Disponibilité *(Avaibility)*

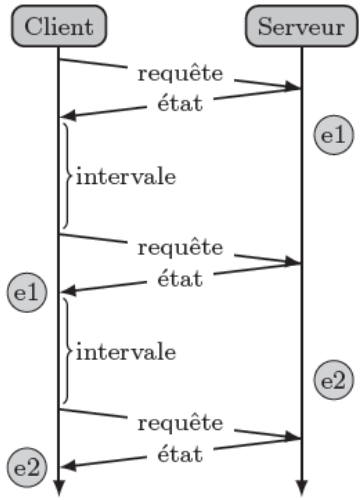
- Motivations
  - Performance
    - Améliorer/garantir les temps de réponse
    - QoS pondérée pour différentes classes d'usagers (gold, silver, …)
    - Elasticité (évolution incrémentale) des ressources CPU (grid/cloud)
  - Disponibilité
    - Tolérance aux pannes
      - Différents niveaux de disponibilité
    - fonctionnement dégradé en cas de panne
      - Le serveur Europe reçoit les requêtes des clients français
    - Failover : Interruption de service
      - Basculement d'une session vers un autre serveur sans interruption de service

Constrained Application Protocol (CoAP)

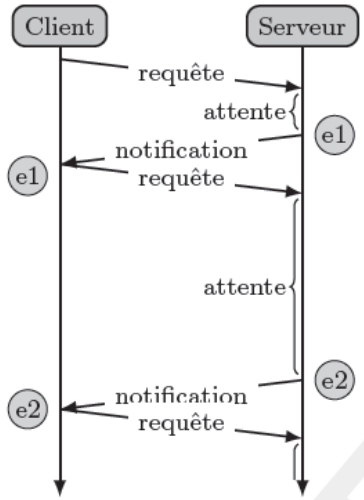
- Motivation : HTTP/REST-like protocol for constrained sensors
  - Typical configuration : 128KB FlashRAM and 4KB RAM
  - Battery consumption (sleep and periodical wakeup)
- Interaction
  - Request-response
  - Subscribe-Notify
- REST (CRUD)
  - Map on HTTP methods : PUT, GET, POST, DELETE
- Resource discovery
  - /profile URI multicast + DISCOVER method
- Protocol binding
  - UDP and UDP Multicast (16-bit sequence number for reliability)
  - Optionally TCP without “stop and wait”
- Caching
  - Important since sleeping mode
  - CoAP proxy (for subscription …)

Notification (ou *Push*) en HTTP

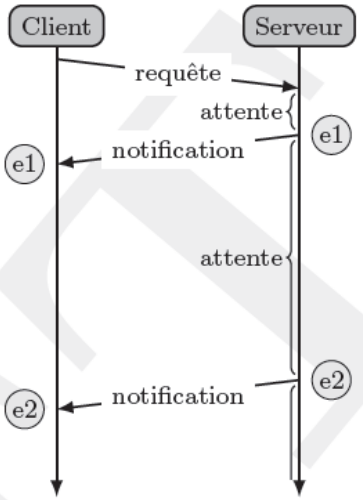
- Motivation
  - Notifier (PUSH) des événements au client
  - « HTTP » full -duplex
- Techniques



(a) *Polling*



(b) *Polling* long



(c) Émission en flux

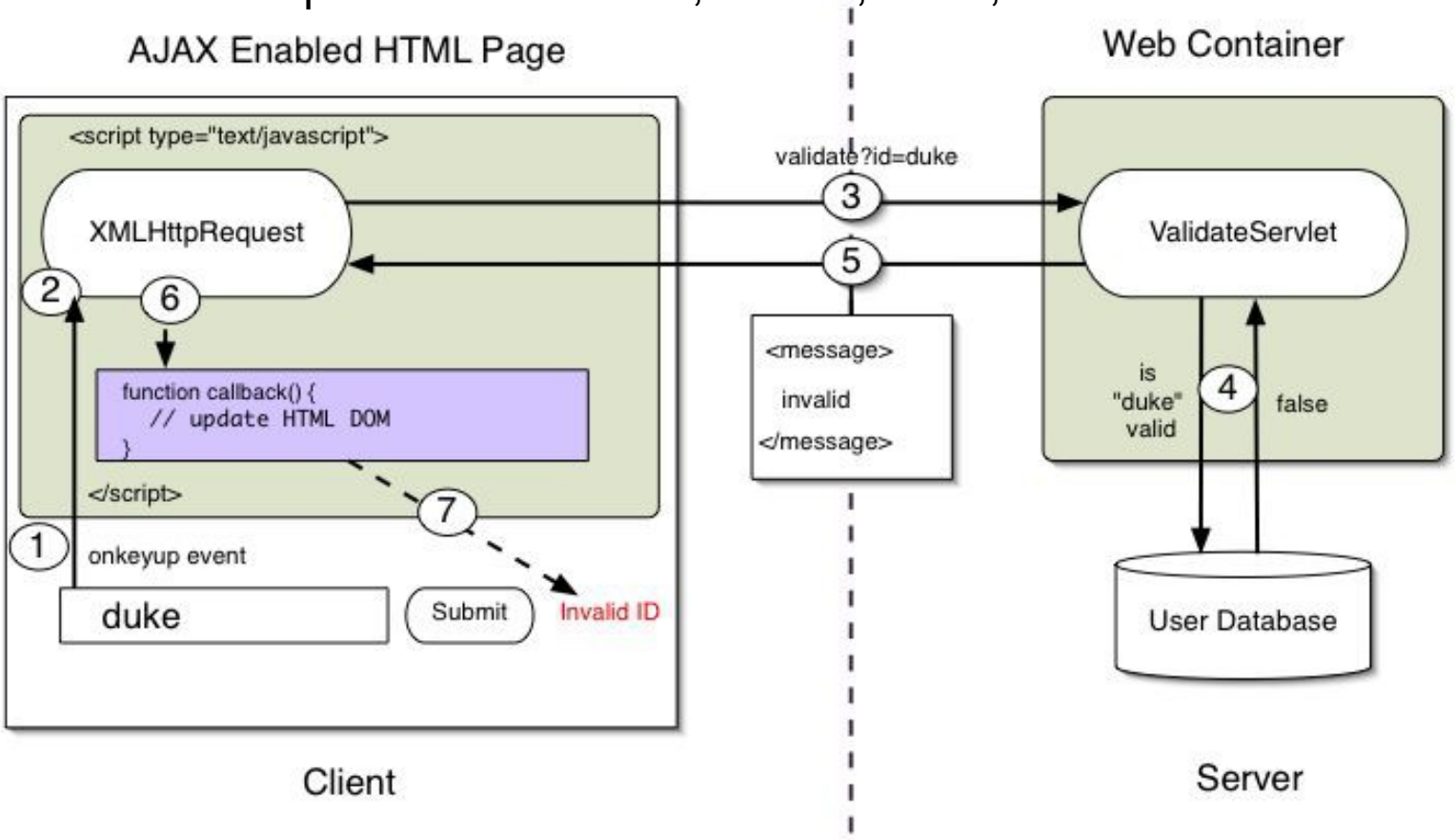
D'après Simon Duquesnoy

- Protocoles
  - Ajax : Objet XMLHttpRequest
  - Comet (Bayeux protocol) : full duplex
  - WebSocket HTML5 : full duplex

AJAX

Asynchronous JavaScript Technology and XML

- Motivation
  - Rich Client (Web 2.0) en JavaScript
- Objet Javascript XMLHttpRequest
  - Réponses en HTML, JSON, XML, ...



WebRTC

Real-time communications on the Web

- Audio/Visioconf
- Helpdesk centers
- Control Command (PTZ)

Built into the browser

- No other installation needed
- No components (e.g. Java applets) to download

JavaScript API is natural and accessible for web developers

- Mobile browser support (phones, tablets)
- Based on open standards
  - Unlike Flash, not under one company's control

Standard specifies codecs, reducing uncertainty

- Voice -– G.711, G.722, ILBC and iSAC
- Video – VP8 (+ H.264? IETF vote results?)

Supports both peer-–to-–peer and intermediated communication

- Everyone is gehng into the WebRTC business

Vendors: Avaya, Cisco, GENBAND...

Open Source: FreeSWITCH, Asteris