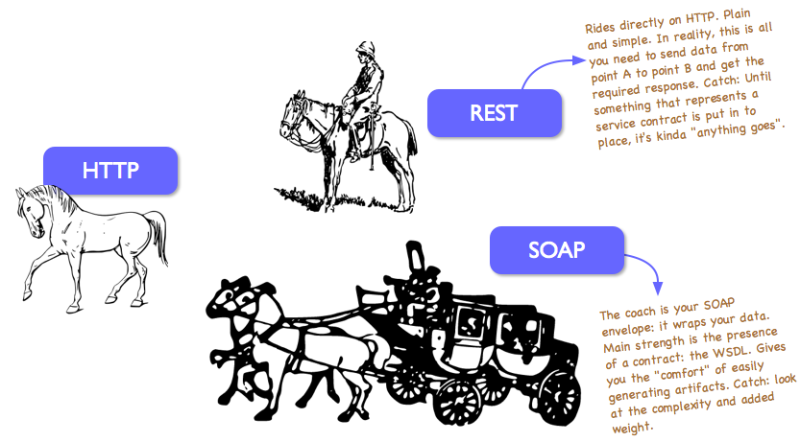
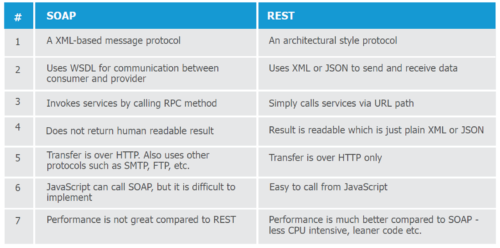
Web services are of two kinds: Simple Object Access Protocol (**SOAP**) and Representational State Transfer (**REST**). **SOAP** defines a standard communication protocol (set of rules) specification for XML-based message exchange. **SOAP** uses different transport protocols, such as HTTP and SMTP.





SOAP (**Simple Object Access Protocol**) and REST (**Representation State Transfer**) both are beautiful in their way. So I am not comparing them. Instead, I am trying to depict the picture, when I preferred to use REST and when SOAP.

**What is payload?**

When data is sent over the Internet, each unit transmitted includes both header information and the actual data being sent. The header identifies the source and destination of the packet, **while the actual data is referred to as the payload**. In general, the payload is the data that is carried on behalf of an application and the data received by the destination system.

Now, for example, I have to send a **Telegram** and we all know that the cost of the telegram will depend on some words.

*So tell me among below mentioned these two messages, which one is cheaper to send?*

<name>Arin</name>

or

"name": "Arin"

I know your answer will be the second one although both representing the same message second one is cheaper regarding cost.

So I am trying to say that, **sending data over the network in JSON format is cheaper than sending it in XML format regarding payload**.

**Here is the first benefit or advantages of REST over SOAP**. SOAP only support XML, but REST supports different format like text, JSON, XML, etc. And we already know, if we use Json then definitely we will be in better place regarding payload.

Now, SOAP supports the only XML, **but it also has its advantages.**

**Really! How?**

SOAP relies on XML in three ways Envelope – that defines what is in the message and how to process it.

A set of encoding rules for data types, and finally the layout of the procedure calls and responses gathered.

This envelope is sent via a transport (HTTP/HTTPS), and an RPC (Remote Procedure Call) is executed, and the envelope is returned with information in an XML formatted document.

The important point is that **one of the advantages of SOAP** is the use of the **“generic” transport**but **REST uses HTTP/HTTPS**. SOAP can use almost any transport to send the request but REST cannot. So here we got an advantage of using SOAP.

As I already mentioned in above paragraph **“REST uses HTTP/HTTPS”**, so go a bit deeper on these words.

When we are talking about REST over HTTP, all security measures applied HTTP are inherited, and this is known as **transport level security** and it secures messages only while **it is inside the wire**but once you delivered it on the other side you don’t know how many stages it will have to go through before reaching the real point where the data will be processed. And of course, all those stages could use something different than HTTP.**So Rest is not safer completely, right?**

But SOAP **supports SSL** just like REST additionally **it also supports WS-Security** which adds some enterprise security features. WS-Security offers protection from the **creation of the message to it’s consumption**. So for transport level security whatever loophole we found that can be prevented using WS-Security.

Apart from that, as **REST is limited by it's HTTP protocol** so it’s transaction support is neither ACID compliant nor can provide two-phase commit across distributed transnational resources.

But SOAP has comprehensive support for both **ACID based transaction management** for short-lived transactions and compensation based transaction management for long-running transactions. It also supports **two-phase commit across distributed resources**.

I am not drawing any conclusion, but I will prefer SOAP-based web service while security, transaction, etc. are the main concerns.

**SOAP – Simple Object Access Protocol.**

* Exposes **Operations** that implements business **Logic.**
* It provides loose coupling to integrate hybrid systems.
* SOAP supports standard error Messaging
* Designed for:
  + Extensible
  + Distributed Computing
* SOAP service very well suited for Enterprise Applications needs & goals:
  + Supports HTTP, SMTP, JMS
  + Enterprise security with WS-Security
  + Supports ACID, Atomic transactions with WS-AT
  + Lot of development tools available
* Supports HHTP POST
* Supports only XML
* Supports stateless &stateful operations
* Strongly typed.
* Supports asynchronous messaging

**REST – Representational State Transfer**

* Exposes **Resources**which represent**Data**
* Use HTTP methods (GET/POST/DELETE)
* Based on simple Point to Point communication over HTTP.
* Stateless communication
* Supports multiple Formats

**Key Advantages of REST?**

* REST can be consumed by any client like web browser, AJAX, JQuery, Javascript
* REST is definitely lightweight
  + Doesn't requires XML parsing
  + Consumes less bandwidth
  + Doesn't require SOAP header for every message.

**Key Challenges with REST?**

* Building a client in REST can be challenging:
  + Can't generate client side artifacts from WSDL as in case of SOAP
  + Need to write raw HTTP calls.
  + Not many IDE available to write REST Clients
* Rest only Support HTTP / HTTPS
* REST is not secure.
  + Parameters are part of the URI
  + No support for acquiring tokens
* RESTFul Services has no contract.
* REST is not reliable
* REST can't be governed as it doesn't have any service registry.

**What's is Verdict for REST vs SOAP?**

Both REST & SOAP are best suited for following respective scenarios

|  |  |
| --- | --- |
| REST | SOAP |
| Best Suited for:   * Web Services. * Smaller size messages (Limited bandwidth) * Limited resources (No XML parsing required) * Exposing data over Internet. * Combining content from different sources / SORs in a web browser. | Best Suited for:   * Enterprise Services   + High availability WS-RM   + Supports transactions WS-AT   + Supports security WS-Security   + Asynchronous Processing   + Contract based development   + Supports interoperability with business applications. |

OAP and REST can't be compared directly, since the first is a protocol (or at least tries to be) and the second is an architectural style. This is probably one of the sources of confusion around it, since people tend to call REST any HTTP API that isn't SOAP.

Pushing things a little and trying to establish a comparison, the main difference between SOAP and REST is the degree of coupling between client and server implementations. A SOAP client works like a custom desktop application, tightly coupled to the server. There's a rigid contract between client and server, and everything is expected to break if either side changes anything. You need constant updates following any change, but it's easier to ascertain if the contract is being followed.

A REST client is more like a browser. It's a generic client that knows how to use a protocol and standardized methods, and an application has to fit inside that. You don't violate the protocol standards by creating extra methods, you leverage on the standard methods and create the actions with them on your media type. If done right, there's less coupling, and changes can be dealt with more gracefully. A client is supposed to enter a REST service with zero knowledge of the API, except for the entry point and the media type. In SOAP, the client needs previous knowledge on everything it will be using, or it won't even begin the interaction. Additionally, a REST client can be extended by code-on-demand supplied by the server itself, the classical example being JavaScript code used to drive the interaction with another service on the client-side.

I think these are the crucial points to understand what REST is about, and how it differs from SOAP:

* REST is protocol independent. It's not coupled to HTTP. Pretty much like you can follow an ftp link on a website, a REST application can use any protocol for which there is a standardized URI scheme.
* REST is not a mapping of CRUD to HTTP methods. Read [this](https://stackoverflow.com/questions/19843480/s3-rest-api-and-post-method/19844272#19844272) answer for a detailed explanation on that.
* REST is as standardized as the parts you're using. Security and authentication in HTTP are standardized, so that's what you use when doing REST over HTTP.
* REST is not REST without [hypermedia](https://stackoverflow.com/a/29586455/1202421) and [HATEOAS](http://en.wikipedia.org/wiki/HATEOAS). This means that a client only knows the entry point URI and the resources are supposed to return links the client should follow. Those fancy documentation generators that give URI patterns for everything you can do in a REST API miss the point completely. They are not only documenting something that's supposed to be following the standard, but when you do that, you're coupling the client to one particular moment in the evolution of the API, and any changes on the API have to be documented and applied, or it will break.
* REST is the architectural style of the web itself. When you enter Stack Overflow, you know what a User, a Question and an Answer are, you know the media types, and the website provides you with the links to them. A REST API has to do the same. If we designed the web the way people think REST should be done, instead of having a home page with links to Questions and Answers, we'd have a static documentation explaining that in order to view a question, you have to take the URI stackoverflow.com/questions/<id>, replace id with the Question.id and paste that on your browser. That's nonsense, but that's what many people think REST is.

This last point can't be emphasized enough. If your clients are building URIs from templates in documentation and not getting links in the resource representations, that's not REST. Roy Fielding, the author of REST, made it clear on this blog post: [REST APIs must be hypertext-driven](http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven).

With the above in mind, you'll realize that while REST might not be restricted to XML, to do it correctly with any other format you'll have to design and standardize some format for your links. Hyperlinks are standard in XML, but not in JSON. There are draft standards for JSON, like [HAL](http://stateless.co/hal_specification.html).

Finally, REST isn't for everyone, and a proof of that is how most people solve their problems very well with the HTTP APIs they mistakenly called REST and never venture beyond that. REST is hard to do sometimes, especially in the beginning, but it pays over time with easier evolution on the server side, and client's resilience to changes. If you need something done quickly and easily, don't bother about getting REST right. It's probably not what you're looking for. If you need something that will have to stay online for years or even decades, then REST is for you.

For example, using the StringBuilder class can boost performance when concatenating many **strings** together in a loop. **String** in **C#** is immutable. If you concatenate it with any **string**, you are actually making a new **string**, that is new**string** object ! But StringBuilder creates **mutable string**.