lightstreamer

# Changing the Web Paradigm

## Moving from traditional Web applications to “Streaming-AJAX”

by Alessandro Alinone

originally written in December 2005
last updated in December 2006

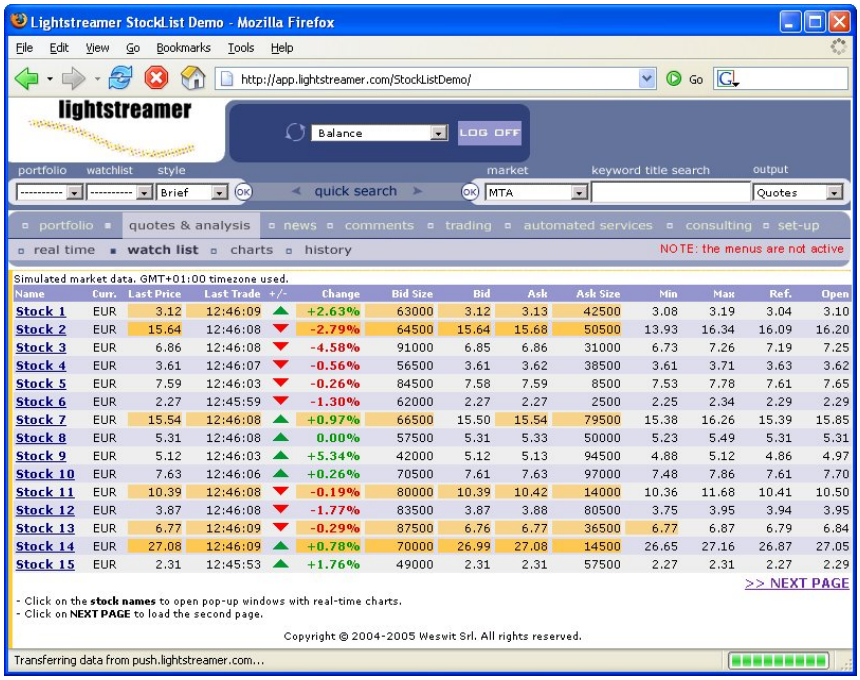
The **classic Web paradigm** (known as “pull”) is **synchronous**: it has the client (browser) solicit data from the server in a synchronous manner. This means that every time the client needs a data update, it has to ask the server expressly to find out if the data has changed and obtain the new value. In other words, for every request from a client there is a corresponding reply from a server. When a Web page is visualized, the data contained within it is static on the user’s browser and is not updated until a page refresh is made (manual or automatic). There are, however, a growing number of applications that necessitate the visualization of real-time data. Current examples are stock prices from on-line trading sites, betting odds from gambling portals, sports results and messages exchanged through online communities. These are just a few cases of systems which, in order to offer the maximum in usability and quality of user experience, require continual updates of the visualized data in the browser page.

In some applications a **polling technique** is used (that is, an automatic page refresh is periodically requested). This type of solution only partially resolves the problem in that:

- The update frequency cannot be high. A synchronous paradigm (request/response) makes it impossible to receive data in real time.
- The occupied network bandwidth is high, because with each response a whole page is transferred, instead of only the changed data.
- The impact on Web server resources is huge, because the server needs to sustain a high load of page requests even though users are inactive.

To fix these problems at the source requires a change in paradigm, from synchronous to **asynchronous**. To guarantee a very low latency between the generation of fresh data and its presentation to the end user within a common browser, a dedicated solution is necessary, namely **Push Technology**. This term was coined in 1996 to refer to any technique that addressed - to a greater or lesser degree and more or less effectively – the challenge of reversing the classic Web model. In the push (or **streaming**) model, the client receives updates in an asynchronous manner at the server's discretion, in the form of a continuous data flow. The client becomes a passive part of the system, receiving updated information as soon as it is available on the server, without having to ask for it periodically. In this sense, e-mail could be considered the oldest and most widespread form of push technology on the Internet. Generally the client subscribes to a certain typology of data (that is,

expresses interest in a certain type of information, known as topic, subject, item, etc.) and then waits to receive the updates in real time as they occur. This paradigm is traditionally known as **publish and subscribe**.




**Lightstreamer** is a push/streaming engine that has been optimized for the real-time distribution of textual data through HTTP connections, based on a pub/sub paradigm. In addition to supporting traditional clients (that is, *thick/desktop* applications), Lightstreamer is also able to stream data to HTML Web pages, without the use of Java applets, ActiveX controls, plug-ins, or Flash components (all of which are supported technologies but are not mandatory to enable *push*). This means that Lightstreamer offers a **zero-client install** solution through its JavaScript API, compatible with practically all browsers in circulation, that is able to "enliven" the Web pages because the displayed data is updated in real time, without any form of reloading. The best way to appreciate this paradigm shift is to view some on-line demos[1].

A streaming connection is used to transmit data updates only when they occur. This means that when there is no new data present at the source, the connection bandwidth is not used. However when an update is generated, it will be instantly delivered to all the browsers that have subscribed to that information.

The Lightstreamer Server has many significant and unique features, for example:

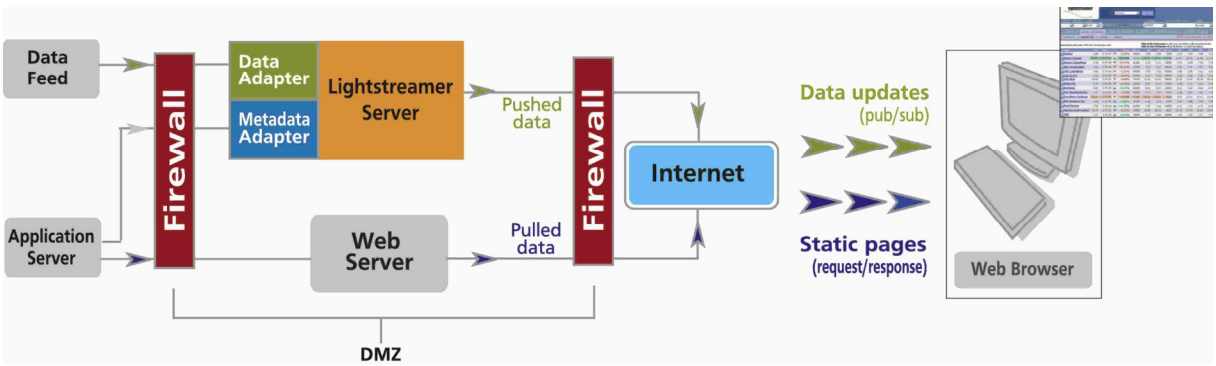
- **Bandwidth and Frequency Control**. Traditionally, pushing real-time data can lead to unpredictable network traffic. This is because each user could subscribe to an arbitrary number of items where the actual update frequency of each subscribed item is changeable. The solution to this issue is to use **dynamic heuristic filtering algorithms** to limit the bandwidth while maintaining the overall data coherency. It is possible to individually assign a maximum bandwidth for each user's streaming channel so that the data will be dynamically filtered to keep the used bandwidth within that allocation (while maintaining guaranteed delivery when required). It is also possible to assign a maximum update frequency for each subscribed data item.

[1] Please visit the demo section on the Lightstreamer site at www.lightstreamer.com/demo.

- **Adaptive Streaming**. Several adaptive mechanisms are employed by Lightstreamer to **throttle the data flow** based on the state of the network. The Server is able to detect possible network congestion and in these cases adaptively change the data transmission rate in such a way that it will never send more data than the network is able to handle at any given instant, avoiding data bursts and data aging.

- **Scalability**. True-push solutions need to maintain at least one open TCP socket for each connected client. Some Web/application servers have been extended to act as streaming engines, but their traditional architecture based on the "one-thread-per-connection" model makes scalability practically impossible for tens of thousands of users. Lightstreamer employs a more suitable architecture, based on a **staged event-driven model** and asynchronous I/O instead of a thread-based model, making it possible to decouple the number of connections that the server can sustain from the number of threads that are employed. The architecture of Lightstreamer Server can sustain thousands of concurrent streaming connections for each CPU. Furthermore the system can scale by the clustering of several servers through a common load balancing appliance.

Lightstreamer Server is implemented in Java and integration with data feeds is performed by writing custom **Java Adapters** or **.NET Adapters**. Therefore the system is open to real-time feeds from any source. Lightstreamer Server does not replace the Web server but sits alongside it in the DMZ. (The Web server delivers the static part of the pages, while Lightstreamer Server delivers the data updates, as shown in the figure below).




## Lightstreamer and AJAX

The advent of Web 2.0, and AJAX in particular, is transforming the Web user's experience to make it very similar to using a desktop application (that is, interactive controls, fast response time, decoupling between user's actions and page loading, etc.). **AJAX**, when used to update information in a Web page, is based on a polling technique, because it does not usually support streaming and needs to make periodic requests to the server to get new data. The Lightstreamer Web Client adds true push to the AJAX paradigm. For this reason the term "**streaming AJAX**" was coined to refer to this new approach[2].

[2] In March 2006, Alex Russell coined the term Comet to refer to AJAX systems that push data to the clients. Lightstreamer can also be classified as a "Comet Server". See http://alex.dojotoolkit.org/?p=545 and http://en.wikipedia.org/wiki/Comet_(programming).

Table 1 summarizes the terms used to describe different application paradigms. Table 2 illustrates the interaction models related to the paradigms. In both tables three tiers (user, browser, server) are used to explain the differences because the type of interaction that occurs between the server and the browser (i.e. the JavaScript engine) can be different from the interaction between the browser and its human user. In particular, a synchronous data delivery with respect to the user's actions means that to update the displayed data the user must take some action or that the user's actions are blocked during the update. A synchronous data delivery with respect to the browser's actions means that even if a user's action is not required, under the hood, the JavaScript code running inside the browser has to issue a request to the server and wait for a response for each update.

| Application Paradigm | Method of sending data with respect to the user's actions | Method of sending data with respect to the browser's actions |
|---|---|---|
| **Traditional Web Application**<br>**à Page Refresh** | **Synchronous** | **Synchronous** |
| **Classic AJAX Application**<br>**à Periodic Polling** | **Asynchronous** | **Synchronous** |
| **Smart AJAX Application (Lightstreamer Application)**<br>**à Smart Polling**<br>**à Asynchronous Polling**<br>**à Comet - Long Poll**<br>**à AJAX Push** | **Asynchronous** | **Partially Asynchronous** |
| **Streaming AJAX Application (Lightstreamer Application)**<br>**à Streaming AJAX**<br>**à True Push/Streaming**<br>**à Comet - Forever Frame**<br>**à Reverse AJAX**<br>(the term is also used for polling) | **Asynchronous** | **Asynchronous** |

**Table 1. Terms used to describe application paradigms.**

**Traditional Web applications** are based on page refreshes (automatic or a user-driven reload action), in the course of which both the user and the browser are "blocked". In other words, the system is completely synchronous. Furthermore, the update frequency that can be reached is very low.

**AJAX applications** have been introduced to avoid blocking the user's actions while retrieving data from the server. But in the classic AJAX model, based on periodic polling, the interaction between the browser and the server is still synchronous, resulting in a waste of bandwidth and in high latency for the data delivery.

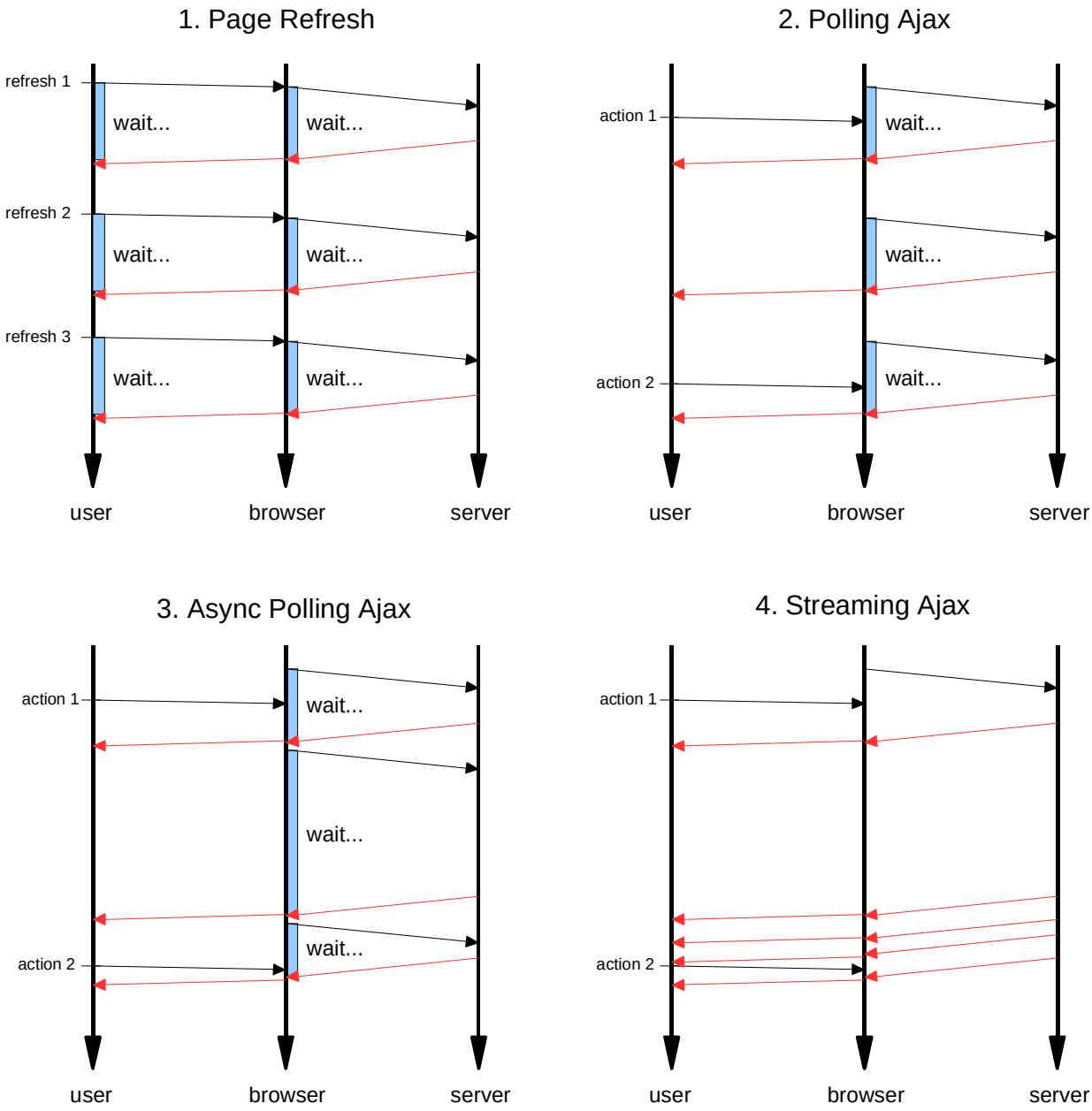



**Table 2. Interaction models related to different application paradigms.**

The **Asynchronous-Polling** paradigm is based on a polling cycle with a variable period. Instead of polling the server at pre-defined times, the client sends a request to the server. It is then up to the server to keep the request pending until fresh data is available, before sending the response. As soon as the client receives the response, it sends a new request. This implies that the polling timing is mainly governed by the server and by the network latency. In this sense it can be defined as **partially asynchronous** with respect to the browser's actions.

Currently, the state-of-the-art paradigm is known as "**Streaming AJAX**" or "**Comet - forever frame**", where true push/streaming is made possible on a very standard Web infrastructure. In this model the data delivery is fully asynchronous, both from the server to the

browser and from the browser to the user's interface. This results in a very high update frequency, with low latency and low bandwidth, leading to an actual real-time system. This paradigm is implemented through a permanent connection from the browser to the server, on top of which the server is able to deliver asynchronous messages adopting a publish/subscribe mechanism. When the browser receives an update through a JavaScript callback function, some code is executed to update the document object model (DOM) of the page in real time, to reflect the data change. Often some graphical effects are employed to call the user's attention to the changing value. It is even possible to plot a live streaming chart in the browser window by using only pure HTML and JavaScript.

The advantages of the completely asynchronous paradigm of Lightstreamer over the other three can be summarized as follows:

- **Zero latency** between the generation of new data and the delivery to the final clients. No need to wait for the next polling cycle to receive fresh data. The asynchronous polling paradigm shares this same benefit, but since a full round trip of request/response is needed for each update, the asynchronous polling mechanism is limited in the maximum frequency allowed for the updates (depending on the network latency). The true push/streaming paradigm allows the highest frequencies for data updates.

- **Reduced bandwidth**. With the true push/streaming paradigm, a permanent streaming connection is kept open for each client. When no fresh data is available, no useless traffic is generated on the connection. Furthermore, the heavy HTTP headers of the round-trip cycles (request/response) of the asynchronous polling are completely avoided.

- **Low load on the infrastructure**. A solution based on polling (whether periodic or asynchronous) needs to generate numerous request/response cycles, which impact the resources of the underlying infrastructure, in particular: the client machine running the browser, the proxy server, the firewall and the server itself. In contrast, the permanent HTTP streaming connection used by the true push/streaming paradigm is efficiently managed by infrastructure, provided that the server is optimized for sustaining a high number of concurrent connections (as is Lightstreamer Server).

True push/streaming paradigm is certainly the best for distributing real-time data. However the asynchronous polling technique should be kept as a backup for situations where some atypical proxy servers block any streaming traffic. What happens in these cases is that a proxy/antivirus fully inspects each Web resource received from the server before sending it to the browser, instead of forwarding it to the client in real time as normal proxies do. No form of streaming is possible (even if the streaming system is based on applets or thick desktop applications). The **Stream-Sense** feature of Lightstreamer automatically detects these situations and transparently switches to the smart polling mode. But even in the polling mode, Lightstreamer retains the unique ability to manage bandwidth, frequency and data filtering.