lightstreamer

WHITE PAPER

the information streaming (R) evolution

Last revised: June 5, 2006

Index

Introduction

This white paper provides a history of push technology, putting Lightstreamer into context and explaining its particular features and potential uses.

The white paper can be consulted by various kinds of readers. It should be borne in mind, however, that sections 1, 2 and 4 are more about the technology, while section 3 discusses applications.

1. What is Push Technology?

The term “Push Technology” was coined in 1996 and immediately became an Internet buzz word. In fact, over time this term has been used to indicate various technological solutions, all of them geared - to a greater or lesser degree and more or less effectively - to reverse the classic web model. The classic model (known as *pull*) 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 to obtain the new value.
The push model, on the other hand, has the client receive updates in an *asynchronous* manner at the server's discretion, generally after expressing interest in a certain type of information. In other words, 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 every so often. In this sense, *e-mail* can be considered the oldest and most widespread form of push technology on the Internet.
According to British linguist Michael Quinion[1], the term may derive from the phrase "push polling" much used by the media during the 1996 US presidential election, which referred to a devious electioneering technique in which canvassers pretended to be conducting a telephone opinion poll, but then used the call to *push* their candidate’s virtues....
Synonyms that have been used over the years for push technology include: *webcasting*, *narrowcasting*, *channeling* and *streaming*.

The history of Push Technology

Basically, it is possible to identify three generations of Push Technology.

First generation: this was born in 1996 and by 1998 it had almost completely disappeared.
The spring of 1996 saw the launch of PointCast, the first push system (in the accepted meaning of the "channels", as explained below). Soon after, at least thirty or more players entered this market niche, including Microsoft and Netscape. IT market analysts expected push technology to become a killer application and that in future it would be the main way of offering information online. This forecast did not come about, not by a long chalk, for the reasons that we shall see. A similar course of events (lots of enthusiasm at the beginning followed by a gradual slide into oblivion) was visible soon after (1998-2000) for another hypothetical killer application: Intelligent Agents...

Second generation: this established itself around the year 2000, but remained extremely fragmented, while also still being immature from a technological point of view.
A small part of the second generation solutions were attempts to launch revamped versions of first generation products with a few technological

[1] http://www.quinion.com/words/turnsofphrase/tp-pus1.htm

improvements. In any case, this was the stage that marked the most tangible sign of the first generation's demise, namely the closure of PointCast (in 2000).
Most second generation push products were developed thanks to the success of online security trading systems, which originated in 1997 and became diffuse world-wide from 2000 onwards. As a result, customized solutions designed to send clients real time stock market information and press agency news flashes became more and more frequent. These solutions not only failed to be consolidated in widely used products, far less in *de facto* standards, but also left various technology problems unresolved.

Third generation: this was inaugurated in 2003 with the official launch of Lightstreamer, conceived and engineered from 2001 onwards.
The fundamental characteristic of third generation push technology (heavily oriented towards so-called *streaming push*) is the definitive resolution of all the technology problems that had been left unresolved by the previous generations, with the introduction of features such as bandwidth and frequency control, the absence of external components to be integrated into front-end web pages and simpler integration with pre-existing sites.
Being based on a solid technological platform, the third generation at long last makes it possible to shift the focus from the technology's problems to how the technology can be applied.

Push architectures

It is possible to distinguish four macro-types of push architecture, which differ in the way that they transfer information from the server to the client.
But first we have to distinguish between two separate stages in the push model: *notification* and *delivery*.
By **notification** we mean a message sent by the server to inform the client that new information is available (in other words, that certain aspects of the site's content have been updated).
By **delivery**, on the other hand, we mean that the new content has actually been transferred from the server to the client.
The four types of push technology are summarized in the following table:

1. **Polling**
   Notification: none — Delivery: synchronous (pull in polling)
2. **Manual smart pull**
   Notification: asynchronous — Delivery: synchronous (manual pull)
3. **Automatic smart pull**
   Notification: asynchronous — Delivery: synchronous (automatic pull)
4. **True push / streaming**
   Notification: none — Delivery: asynchronous

In the case of *polling*, we talk about “simulated push”. There is no notification of updates so the client has to ask the server periodically for information to see if any of it has been changed. This technique not only consumes a great deal more bandwidth during the connection, but also fails to ensure that the data is received in real time; nor is it able to support medium/high update frequencies.

Smart pull consists of the server sending just a notification of updates on an asynchronous basis. It is then up to the client to ask for the new information from the server on an asynchronous basis (i.e. pull). Uploading the new data can be automatic (the client requests the update as soon as notification arrives) or manual (the user decides to ask for the new data some time after having received notification). The following is an example of manual smart pull. The user receives a newsletter via e-mail containing links to web pages with the new information that has just been published; the user clicks on a link and opens a browser window where the contents can be uploaded. Notification of the content updates is therefore asynchronous (by e-mail), while delivery is synchronous (by pulling a web page). Another example of manual smart pull are those *Instant Messengers* which notify in an asynchronous way the presence of new e-mail in your inbox; it is then up to the user to connect to the web mail page to pull down the messages.

Smart pull has a lower impact on bandwidth consumption compared with polling, but again this technique does not allow you to send information in real time.

True push, also known as *streaming push* or more simply *streaming*, does not use a preliminary notification phase, but has the server send off an update directly to the client on an asynchronous basis as soon as the new information is available. This is the only method that achieves a full level of real-time communication. In fact, the server sends off a continuous flow of updates to the client computer, making them immediately visible to the user.

The first generation of push was based prevalently on polling and smart pull. The second generation was largely based on polling and to a lesser extent on true push. The third generation, on the other hand, uses exclusively true push.

Push applications

There are various kinds of push applications related to the generations of push technology and the various types of architecture described above. The main ones are as follows:

Channels: This is an application that is strongly linked to the first generation of push technology. The user registers with some "information channels" and then receives their contents, which are visualized by

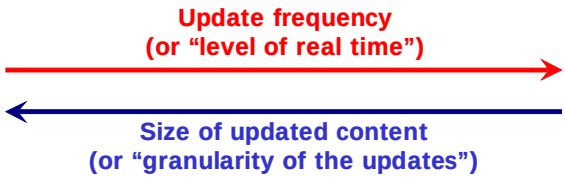
dedicated client terminals, browsers or by special screen savers. The system is based on polling or on smart pull.

Offline browsing: the user indicates which sites they are interested in and the system periodically synchronizes a local cache in the user's PC, making it possible to surf these sites offline. The system is based on polling.

Multimedia streaming: this can be considered a form of push. The user receives a stream of audio and video data, which special software decodes and plays in real time.

Text streaming: The page visualized by the user's browser does not get reloaded, but the data contained in it gets updated in real time (the granularity of the updates is that of the individual "cell").

Generally speaking, the greater the volume of content being updated (i.e. the less fine the granularity of the updates), the lower the frequency of the updates (i.e. the system's level of real time) and vice versa.



For example, systems based on channels have as the granularity of their updates a whole web page, or even a series of pages, with updates being made approximately once a day. Text streaming systems, on the other hand, have an update granularity which is that of an individual cell on the page being visualized, so the frequency of updates can be higher than one a second.

Why is a third generation of push technology needed?

Various analyses have been made into the reasons for the sudden demise of push technology's first generation. To a certain extent, the reasons were technological. To start with, early push systems not only saturated the bandwidth of the individual user's Internet connection, but also caused bottlenecks in Internet providers' backbones... Moreover, users did not appreciate the fact that they had to install special software on their PCs. But the main reason for its demise was probably application-related. There has never been a real need to get to the office in the morning and find your desktop full of new web pages, already downloaded, which you are never going to have time to read. Someone likened the first generation of push technology to immense piles of newspapers being left on your doorstep in the morning...

But if the first generation failed, the second still isn't widely used, consolidated or standardized. In this case, the problems were prevalently technological (indeed they still are, considering that the installed base of second generation push systems is still a good deal higher than those of the third generation). Apart from the fact that many still use simulated push instead of true push, the main problems are of the following type. There are cases in which push updates are unable to get through network structures containing proxies, firewalls and NAT systems. In addition, the Internet connection bandwidth of the user's PC still risks being saturated without warning. What happens then is that the user cannot perform other tasks online (such as consult the mail or surf other websites) while a push session is in progress. However, the user's surfing may be blocked not only by the modem's bandwidth being saturated, but also by the push system exhausting the pool of browser connections. Indeed, in most second generation push systems, each pop-up window or frame showing push data uses up one browser connection, and browser connections are a limited resource. Generally speaking, once four connections are being used simultaneously, browsers get blocked (i.e. they refuse to open other connections) until at least one of them gets freed up.

Normally, second generation systems are unable to handle Internet congestions very well. This means that if for a few seconds the quality of the connection between the client and the server deteriorates, the push server continues to generate information at the same rate, with the result that the client then receives a mass of old data.

But the most obvious problem of second generation push technology is without a doubt the fact that a Java applet, an ActiveX control or a plug-in always has to be downloaded to the user's browser, with the consequences that we will discuss later on.

2. Lightstreamer: the revolution of push technology

The Lightstreamer solution was launched early in 2003 (whilst the inception of this project dates back to 2000), having achieved all of the objectives to overcome the technological limitations of second generation streaming/push systems.

True streaming/push in pure HTML

Lightstreamer is able to update the information shown on an HTML page without having to reload it, and above all, without having to add anything "extraneous", such as a Java applets or plug-ins of any kind. Lightstreamer makes it possibile to develop ***live* Rich Internet Applications** (**RIA**) in pure HTML/JavaScript without using Java or Flash.

This type of solution has all sorts of advantages:

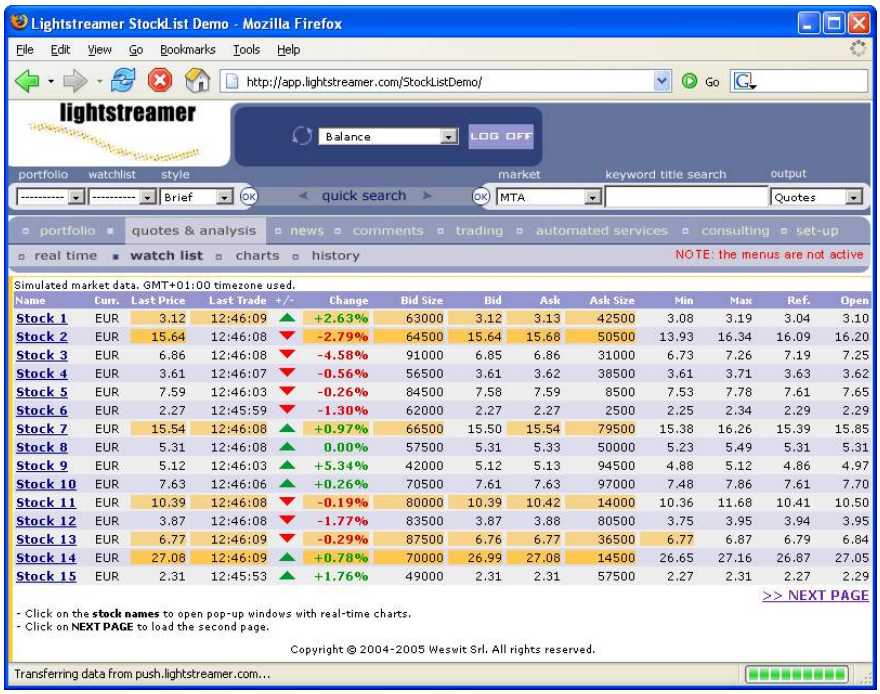
- **Immediate start**. There is no need to wait for a download or for the initialization of an applet or an ActiveX control. Page updates are immediate, without any initial latency.
- **No issues with the Java virtual machine (JVM)**. The lack of applets in Lightstreamer means that there are no compatibility problems with the Java virtual machine in the user's browser. In fact, there is longstanding litigation between Microsoft and Sun Microsystems regarding the presence or otherwise of a JVM in Internet Explorer. So there are numerous versions of JVMs and in certain cases (the latest versions of Windows operating systems), there could be no JVM at all. Lightstreamer avoids these problems totally, because it does not use or require Java on the client side.
- **No security risk**. The presence of an external element on a page, however much it is digitally signed and guaranteed as secure, still leaves room for possible security gaps in the system. Lightstreamer's front-end does not introduce any variance in the level of security compared with that already adopted for a normal pull-type web page.
- **Layout homogeneity and ease of maintenance**. Another huge problem relating to the presence of external elements on the page is the fact that they do not visualize data in HTML, but use proprietary technologies. For example, the use of an applet to visualize data on a push basis forces the applet's developer to simulate the HTML layout of the page containing the applet, using primitives or Java's graphic libraries. This is already a complex and costly exercise in itself. If then the layout of the site were to be personalized by the user (different color schemes, etc) or in any case subject to maintenance, the source code of the applet would have to be adjusted to redesign the front-end to make it compatible with the pages of the website. In fact the site's style sheets are not applicable to external elements. The applet problem is identical with ActiveX controls and plug-ins. Lightstreamer, on the other hand, visualizes push data directly in HTML. Maintenance of the push pages is therefore identical to that of the pull pages, which

means that it does not need programming skills nor the time and cost of a programmer.

How true streaming/push works in HTML

Lightstreamer updates the information on an HTML page by means of a suitable combination of JavaScript, DHTML, CSS and DOM technologies. The most appropriate technologies have been chosen case by case to provide total compatibility with different browsers and to guarantee the best possible performance in any situation. Lightstreamer's push update system therefore functions with *Microsoft Internet Explorer* v.4 onwards, *Netscape* v.4 onwards, with all variants of *Mozilla* and *Firefox*, with *Opera*, *Safari* and many other less diffuse browsers[2].

The site's push pages get updated by means of JavaScript streaming techniques. This involves solutions that found some space in IT sector literature around the year 2000, but they were never widely used because of a series of technological problems that were never completely resolved in an organic way. Lightstreamer has brought the management of JavaScript streaming to a level that is state-of-the-art, giving it a degree of reliability and robustness that is at long last suitable for **production and mission critical applications**.



Only the data that has effectively been changed gets updated on the HTML page. Any change in a figure is accompanied by a graphic effect that can be personalized by the integrator; usually this means temporarily lighting up the cell containing the new data.

[2] Please see www.lightstreamer.com/compatibility.htm for an up-to-date list of compatible browsers.

The high quality of these visualization techniques, together with Lightstreamer's other features which we will discuss later on, means that push technology is at long last a real, simple and complete experience, within the reach of almost any web site. In fact, visualizing a push page is now identical to visualizing a normal pull page. It is as though a **whole new dimension had been added to the experience of web surfing**. Imagine an HTML page that loads itself and immediately starts updating its own content "**live**", without any prerequisite on the part of the system.
To see some online live demos of Lightstreamer technology, just go to the site www.lightstreamer.com and click on the **DEMOS** menu. You will see how with Lightstreamer it is even possible to **plot real-time graphic charts** with pure HTML/JavaScript.

As explained later on, Lightstreamer is also the ideal solution to feed information to traditional application clients on a push basis (written in any programming language: Java, C++, C#, Visual Basic, Flash, etc.).

Lightstreamer and AJAX

AJAX or **Asynchronous JavaScript and XML** is a term describing a web development technique for creating interactive web applications using a combination of HTML (or XHTML), CSS, DOM, XML, XSLT and the XMLHttpRequest object. With respect to traditional web applications, AJAX applications can send requests to the web server to retrieve only the data that is needed instead of a full HTML page, usually using SOAP or some other XML-based web services dialect, and using JavaScript in the client to process the web server response. This results in more responsive applications, because of the reduction in the amount of data interchanged between the web browser and web server to display an up-to-date page.

"Classic" AJAX, when used to update the data displayed in a page, **is a polling technique** (see "Push architectures" on page 6). It does not support streaming and needs to periodically make an enquiry to the server both to know if fresh data is available and then to retrieve it. The following table highlights the differences between four paradigms. The columns refer to the methods of sending data with respect to the user's and the browser's actions. Synchronous with respect to the user's actions means that to update the displayed data the user must take some action or (if automatic reload is used) the user actions are blocked during the update. Synchronous 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 user's actions	Method of sending data with respect to browser's actions
Traditional Web Application --> **Page Refresh**	Synchronous	Synchronous
Classic AJAX Application --> **Periodic Polling**	**Asynchronous**	Synchronous
Smart AJAX Application and Lightstreamer Application --> **Smart Polling** --> **Asynchronous Polling** --> **Long Poll**[3]	**Asynchronous**	**Partially Asynchronous**
Lightstreamer Application --> **Streaming AJAX** --> **True Push/Streaming** --> **Forever Frame**[3]	**Asynchronous**	**Asynchronous**

Classic AJAX applications can update the data displayed in a page without the user having to explicitly request it (the data delivery is asynchronous with respect to the user). But the AJAX engine has to retrieve the data from the server in a synchronous fashion. The **asynchronous-polling** paradigm is based on a polling cycle with a variable period. Instead of polling the server at predefined times, the client sends a request to the server. It's 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. On the other hand, Lightstreamer applications are fully asynchronous, because when new data is available, it is pushed *by* the server *to* the client. For this reason, Lightstreamer's paradigm could be defined as ***streaming AJAX***. The major advantages of a full asynchronous approach based on streaming/push are:

§ **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

[3] In March 2006, Alex Russell coined the term *Comet* to refer to AJAX systems that are able to push data to the client (see http://alex.dojotoolkit.org/?p=545 and http://alex.dojotoolkit.org/?p=547). So Lightstreamer can be defined as a "Comet Server". Comet uses the terms "Long Poll" and "Forever Frame" to refers to these two techniques.

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.
- **Very low load on the infrastructure**. An approach based on polling (whether periodic or asynchronous) needs to generate numerous request/response cycles which impact on the resources of the underlying infrastructure, in particular: the client machine running the browser, the proxy server, the firewall and server itself. On the contrary, 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 Lightstreamer Server).

The push/streaming paradigm is certainly the best one to distribute real-time data. However there exists some antivirus software installed on proxy servers that blocks any form of streaming. What happens is that this type of 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. In this case no form of streaming is possible (even if the streaming system is based on applets or thick desktop applications). The **StreamSense** feature of Lightstreamer automatically detects these situations and transparently switches to the **Smart Polling** mode. But even in the polling mode, Lightstreamer keeps the unique ability to manage bandwidth, frequency and data filtering.

Lightstreamer and Jabber/XMPP

The term **Jabber** is widely used to refer to a set of open protocols for streaming XML elements between any two points on a network, and to the technologies built using those protocols. The **Extensible Messaging and Presence Protocol** (**XMPP**) is the IETF's formalization of the base XML streaming protocols for instant messaging and presence developed within the Jabber community. XMPP is usually implemented via a client-server architecture wherein a client utilizing XMPP accesses a server over a TCP connection on port 5222. In addition to the base protocols, there exist several extensions specified in the “JEP” series, including two methods for using HTTP connections instead of plain TCP sockets: **HTTP-Polling** (JEP-0025) and **HTTP-Binding** (JEP-0124). Anyway, none of these extensions support asynchronous streaming, since they are based on periodic requests from the client to poll the server for incoming “stanzas” (i.e. XML discrete semantic units of structured information). This implies that Jabber/XMPP with its standard extensions is not able to stream data with zero latency to pure HTML/JavaScript clients.

Lightstreamer can be used as a gateway for Jabber/XMPP servers to implement a real streaming mechanisms for thin clients based on HTTP, with three main advantages:

- **Pure HTML/JavaScript clients for Jabber** can be developed, without using applets, plug-ins or Flash components.
- **Real streaming** is made available on very common network infrastructures, without reconfiguring firewalls or proxies.
- **Bandwidth usage is optimized** due to the efficient network protocol of Lightstreamer that is not based on verbose XML.

To use Lightstreamer as a gateway for Jabber/XMPP servers it is enough to develop a XMPP Data Adapter for Lightstreamer.

The architecture of Lightstreamer Server

Lightstreamer Server can be installed as a *cluster*. Each cluster node consists of a Java process, which includes three modules:

- **The Kernel of Lightstreamer Server**: This is the real engine of the push system as it maintains the HTTP/HTTPS connections with the clients and distributes the data, filtering it on the basis of the allocated bandwidth and frequency. Lightstreamer Server is a stand-alone process, which does not rely on any web server, servlet container or application server. This approach was taken in order to have direct control over the operating system's TCP/IP stack and to optimize the data transmission as much as possible. The Kernel of Lightstreamer Server has the task to distribute the data to the clients in a very reliable and efficient way, offering unique features like Bandwidth & Frequency Control and Adaptive Streaming.
- **The Data Adapter**: This is a plug-in module which interfaces Lightstreamer with the data source (or "feed") to be integrated. The Data Adapter receives a flow of data from the back-end systems (information provider, data feed, database, etc.) and makes it available to Lightstreamer Server for controlled delivery to individual users. The Data Adapter can use any technology to integrate itself with the data feed. However, it is preferable to use middleware equipped with asynchronous paradigms, such as message-oriented systems (JMS, RendezVous, MQ), so as not to break the asynchronous chain that goes from the feed to the user's browser. In any case, it is also possible to use polling techniques to refresh the data (e.g. reading from file or from database).
- **The Metadata Adapter**: This is a plug-in module which provides Lightstreamer Server with the *metadata* of the push sessions. In particular, given that Lightstreamer Server interfaces directly with the Internet, it has to be able to authenticate users. The Kernel therefore uses the Metadata Adapter to validate the credentials received. This Adapter also manages the description of the information categories handled, the push message schema, user bandwidth and frequency policies and any kind of information that does not consist of real push data, but data of a higher level (i.e. metadata).

The architecture of Lightstreamer makes it possible to handle an arbitrary number of different Data Adapters and Metadata Adapters. In this way it is possible to integrate heterogeneous sources of information while maintaining a single point of access to the streaming/push channel.

Bandwidth control

For each user, Lightstreamer makes it possibile to allocate a maximum bandwidth that is dedicated to the streaming channel. For example, if it is required that a certain user cannot exceed a bandwidth of 10 kbps, Lightstreamer will filter the data in such a way as to ensure that the streaming connection with that user always remains below 10 kbps..
The system is also able to allocate a maximum update frequency for each user/item combination. For example, it is possible to configure the profile of a certain user so that they do not receive more than 2 updates per second for a certain subscribed item (i.e. a piece of data, such as the price of Sun Microsystems stock).

Filtering the data makes sense not only to limit the bandwidth occupied by the streaming connection, but also to improve the quality of data usage on the part of the end user. The human interface can absorb data by a combination of the human eye and brain, which are incapable of perceiving data changes of more than a few updates per second. Sending 30 updates a second would be a waste of bandwidth and computing resources, as they could never be taken in by a human user. However, Lightstreamer can also feed automated (as opposed to human) processes. In this case, it is possible to bypass the filtering mechanisms.

Filtering data is a possibility offered by the intrinsic nature of many kinds of information. All information sources that over time generate updated versions of the same data, more or less frequently, can quite easily be subjected to filtering algorithms.

Examples of filterable data feeds:

§ **Stock exchange prices.** The price of a stock ("last price") varies continuously during the course of a stock market session. Once the market has generated a new price, the old one automatically becomes obsolete. So in the case where prices are generated with a very high frequency, one could decide not to communicate each and every variation. It is obviously indispensable to maintain the consistency of the data presented on a page, despite the filtering. For example, if on the same page there is the price of a covered warrant whose price gets updated once every hour, together with the price of a stock listed on the NASDAQ, which generates 20 updates a second, it is obviously

unthinkable to lose the one update of the first in order to communicate more updates of the second. Lightstreamer's powerful heuristic mechanisms ensure that whenever one takes a "snapshot" of a page, all of the data on it are absolutely coherent and consistent.

§ **Measurements by a probe.** A probe of any sort (e.g. a physical probe to measure temperature or a software probe for network management) produces a whole series of measurements which are samples of the quantity that the probe is designed to monitor. These samples can be subsequently re-sampled to reduce their frequency even further.

Lightstreamer manages the filterable data feeds by means of a subscription method called ***MERGE***.

However, there are data feeds that should not be filtered. These are feeds that do not produce data that replaces the previous data, but which produce data that is displayed alongside the previous data. A typical example of this kind of data feed is:

§ **News headlines.** Press agencies generate fresh news items throughout the day. Typically, a real-time news visualization system shows a series of headlines in chronological order (from the latest to the oldest) which runs on receipt of each new headline. So when a new item of news is received, it does not eliminate the previous one; it gets listed alongside it.

Lightstreamer manages the non-filterable data feeds by means of a subscription mode called ***DISTINCT***. In this case, Lightstreamer delivers each single update. Any bursts in the generation of new updates get absorbed by spreading out their dispatch to clients over time, within certain limits.

In addition to the MERGE and DISTINCT modes, Lightstreamer also offers another two modes of subscribing push data:

§ ***RAW.*** The flow of information received from the data feed is sent 'as is' to the clients, complying solely with the bandwidth restrictions. It is a suitable method for Lightstreamer to feed automatic processes that require all of the updates (e.g. a system for recording the history of a series of figures in a data base).

§ ***COMMAND.*** This makes it possible to handle the so-called "**meta-push**". There are data feeds that deliver two or more levels of updates. For example, the classic list of today's 10 best shares on the Stock Exchange gets updated in real time at two levels: at any moment in time, the composition of the list can change (i.e. which shares are included), as can the details of each item on the list (individual share prices). Lightstreamer's COMMAND mode makes it possible to manage meta-push easily and efficiently. Also in COMMAND mode it is possible to carry out automatic server side compensation to reduce the number of updates being sent. Going back to our previous example of the 10 best shares, if within the space of a tenth of a second a share entered and then exited the list, the server could decide not to send customers either of the two updates, in order to save on bandwidth. These sophisticated mechanisms can be

controlled by means of a whole series of parameters, which make them suitable for all kinds of filtering needs. For live examples of COMMAND subscription mode, please see the Portfolio and Paged Portfolio online demos on www.lightstreamer.com web site.

The **advantages of bandwidth control** are obvious, both for the server and for the client:

- **For the server**, bandwidth control makes it possible to size the Internet connection required simply and accurately. If each user is allocated 5 kbps and maximum usage at any one time is expected to be 1000 simultaneous push users, then the maximum connectivity that is needed at peak times is 5 Mbps. Without bandwidth control, it becomes very difficult to predict how much connectivity is needed, with the risk of saturating the network and causing other services to malfunction as well.
- **For the client**, bandwidth control makes it possible to avoid saturating the bandwidth of the user's modem. Traditional push connections through narrow band equipment (e.g. analog modems or GSM/GPRS mobile networks) risk rapidly saturating the available bandwidth and blocking the user's access to other services. With Lightstreamer, 4 kbps can be allocated to the push channel on a GSM mobile phone, leaving about another 5 kbps free to access other Internet services. In other words, bandwidth control allows *partitioning* of the client's bandwidth among various applications.

Adaptive streaming

Internet connections are often unable to offer a guaranteed bandwidth. This means that the ability to allocate a maximum bandwidth to each user may not be sufficient to ensure that the push connection is managed in the best way possible. The system has to be able to take into account exactly how much bandwidth is available at any moment in time.
The sudden bandwidth bottlenecks that take place every so often on the Internet may be due to congestions on intermediate links of the network, which cause TCP packets to be lost and subsequently retransmitted (the reliability mechanisms of TCP protocol ensure that any packet that does not reach its destination gets sent another time). Lightstreamer automatically spots situations of congestion on the Internet, heuristically slowing down or suspending the dispatch of data until the connection is fully available again. This means that if at any time the bandwidth actually available is less than what was allocated and required, Lightstreamer uses its own filtering mechanisms to modulate the sending of data with a more suitable bandwidth. The advantage of this is that when the channel becomes available again, the user does not receive a burst of obsolete updates, but starts seeing the new data immediately (in other words, so-called "data aging" is avoided). If the server continued to produce new data without it being able to reach the client in real time, it would create a tailback of events that would get longer and longer. And this could not only

lead to saturation of the system, but would also considerably reduce the level of real-time information received by the user.

The advantage of adaptive streaming can be seen particularly well on connections that are notoriously unstable and variable, such as those on the **GPRS network**. Let's assume that the user starts a push session on their GPRS smartphone (using one of the many mobile clients made for Lightstreamer). If at a certain point the GPRS signal deteriorates and the effective bandwidth narrows, or network coverage gets interrupted for a short period of time, Lightstreamer Server understands what is happening and adapts the streaming in a dynamic way to the new network conditions, even going so far as to suspend the sending of data in the event that no bandwidth is available at all. As soon as the connection quality improves, Lightstreamer Server starts sending out new push data again without recovering the history of data that at this stage is obsolete (of course only in MERGE mode while always maintaining the underlying consistency of the client's status).

Lightstreamer also has direct control over the composition of the TCP packets. Instead of delegating the aggregation of data in packets to the operating system using the *Nagle algorithm*, Lightstreamer Server itself decides on each occasion the optimum composition of each TCP packet, with the objective/trade-off of reducing the waiting time before data is sent off and at the same time minimizing the number of packets dispatched. In this way it is possible to improve the reliability of the bandwidth control mechanisms, increase the system's real-time operating capacity and improve the overall efficiency of data transmission.

Lightstreamer's bandwidth control and adaptive streaming make it possible to obtain quality push sessions with **less than 5 Kbps**.

HTTP(S) connections

Each Lightstreamer client typically opens **a single permanent connection** with Lightstreamer Server, on which the push updates relating to an arbitrary number of items, frames and windows travel by means of *multiplexing* techniques. The approach has various advantages:

- The system's response time is better as subscriptions of new data trigger off the immediate dispatch of the updates on a connection that has already been set up.
- The reduction in the number of connections that are needed ties up fewer system resources both on the client's side and on the server's.
- In the case where the client is a web browser, it is of fundamental importance to minimize the number of connections being used. Normally, browsers establish a maximum number of simultaneous connections that are allocated in a pool. If the push system uses more

than one permanent connection, there is a risk that the user's surfing on this and other sites will be completely blocked[4].

Lightstreamer uses exclusively standard **HTTP** or **HTTPS** connections to feed data to its clients. This makes it possible to go through any kind of **proxy** or **firewall** placed between the client and the server, without any sort of reconfiguration. If the user is able to visualize a normal website, they will undoubtedly be able to use a push site fed by Lightstreamer.

Lightstreamer Server uses point-to-point connections with its own clients. On the other hand, it does not use IP multicasting systems (which are certainly more appropriate for audio and video streaming) as these do not permit dynamic modulation of the bandwidth user by user. IP multicasting systems also have little support outside local networks and through common Internet providers.

Clustering

Lightstreamer Server is completely **scalable**. It is possible to create a **cluster** using normal **web load balancing appliances**. In this way, whenever there is an increase in the number of users, it is sufficient to add machines to the cluster to scale the system in a transparent manner, without interruption.
Clustering also makes it possible to handle **fail-over**. If one of the machines in the cluster breaks down, the streaming connections set up with certain clients get interrupted and these clients immediately try to reconnect with the cluster. At this point, the web load balancing system directs the new connections to another machine in the cluster, which will continue to serve the users in lieu of the machine that has broken down.

A high performance server

Lightstreamer Server is a **real-time system** implemented in **Java**, which has been subjected to years of continuous optimization, refactoring and fine-tuning. In order to guarantee the maximum level of performance, Lightstreamer Server has been developed as a stand-alone process which does not rely on an underlying web server or application server. In this way it is possible to have direct control over the TCP/IP layer of the system, as well as being able to implement mechanisms to control concurrence that are perfectly consistent with the specific needs of such a particular server.
Starting from version 2.1, Lightstreamer Server is based on a **staged event-driven architecture**, that allows impressive performance and the ability to sustain very high loads.
The implementation leverages several advanced mechanisms, such as:

- **Stage and queues** to manage events.

[4] It is possible to see "paradoxical" systems of online trading where the user opens four pop-up windows with a 5-level book in push on five different stock market securities. But when the user tries to input a purchase/sale order they can't, because all of the browser's pool connections are tied up to provide push data...

- **Multiple read/write locks** for concurrence management.
- **Thread pooling** and priority management of the threads.
- **Load shedding** mechanism to maintain constant throughput.
- **Object pooling** in memory to minimize garbage collection.
- **Multi-stage data filtering** to give the system maximum scalability.
- **Java NIO** libraries to decrease the number of used threads.

Lightstreamer Server is compatible with version 1.4 or higher of the Sun Java virtual machine and with other JVMs compliant with Sun's standards.
On a monoprocessor machine, the benchmarks have demonstrated that Lightstreamer Server is able to support thousands of simultaneous streaming connections.

Lightstreamer administration

Lightstreamer Server has a **logging** system with dynamically modifiable trace levels. It is therefore possible to trace each single item of data that gets sent to each client in the test phases of the Adapters and of the overall system.
A series of statistics are also exported and updated in real time, covering various aspects of how the system functions, such as:

- resources in use (in terms of memory, threads, pool, etc.)
- open sessions
- generated events
- system anomalies
- bandwidth allocated and utilized
- per-user statistics (filtered events, bandwidth saturation, etc.).

This information is made available in two ways:

- A monitoring console in **HTML** is provided, fed in push by Lightstreamer itself, which shows in real time constantly updated statistics for the system. More generally, there is an **Adapter** within Lightstreamer which allows clients to subscribe to the stream of statistical data provided by the server.
- This information is also exported via the **JMX** standard, which permits integration with various kinds of administration consoles. JMX (*Java Management eXtensions*) is a widespread specification that provides a management architecture and API to allow any Java based technology or accessible resource to be inherently manageable. Lightstreamer 2.3 includes Sun's JMX Reference Implementation and JMX Remote API. Due to this integration, the system administrator can now remotely monitor the quality of service of the Lightstreamer live data channels, together with the management of many other different aspects of the system. Any JMX-compliant management console can communicate with the JMX 'agent' included in Lightstreamer Server. A bridge for the MBean Server included in BEA WebLogic 8.1 is also available.

3. Potential uses of Lightstreamer

Lightstreamer makes available, simply and efficiently, **a new and powerful model for making use of real-time information**, which can be sent through various kinds of channels to clients using different technologies. Numerous applications can benefit from Lightstreamer and combining this technology with pre-existing systems is straightforward and fast.

Heterogeneous clients

Lightstreamer is able to feed various types of client in true-push mode:

- **Web browsers** (pure HTML/JavaScript pages)
- **Desktop applications** (thin or thick application clients written in Java, Visual Basic, C/C++, C#, Flash, etc.)
- **Excel spreadsheets** and **DDE clients**
- **Midlets** (Java J2ME applications for mobile devices)
- **Micro-browsers** for smartphones and PDAs.

Web browsers

As explained in the previous paragraph, Lightstreamer is able to feed a simple **HTML page** in streaming/push mode without any need to install Java applets, ActiveX controls or plug-ins on the browser.
In this scenario, the "client" consists of special **JavaScript libraries** which handle the network connections and make it possible to update the data on the web pages in an extremely **efficient** manner.
The **visual quality** of the updates is high (lighting and coloring effects of the cells containing the updated figures are available).
The JavaScript code contained in the Lightstreamer libraries is such as to ensure **compatibility** with all main browsers available on the market.

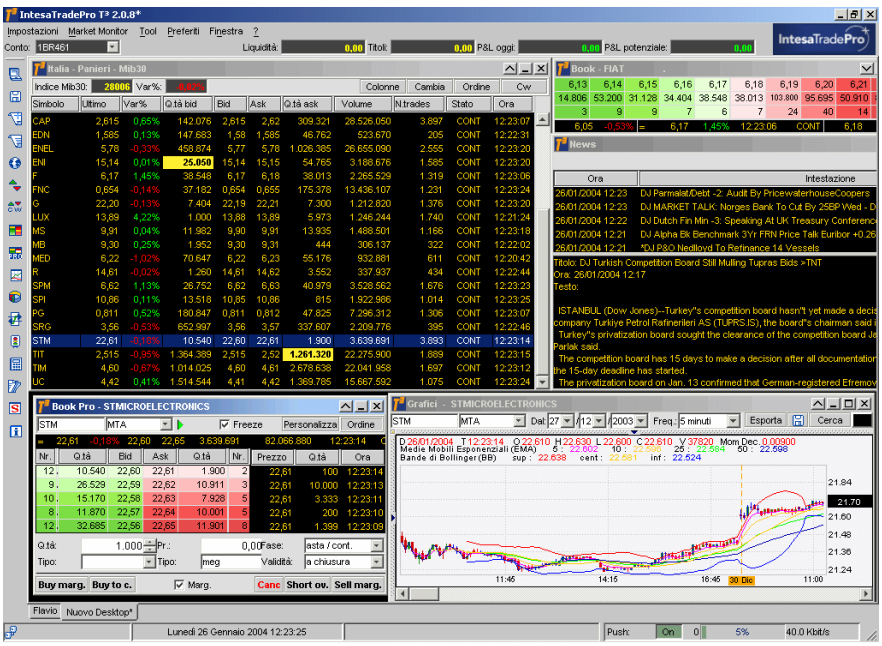
The JavaScript libraries of Lightstreamer permit effects such as:

- § updating of the values displayed in the page
- § dynamic coloring and style changing of values through style sheets
- § on-the-fly creation of new rows in scrolling tables
- § client-side dynamic sorting of tables
- § client-side paging
- § plotting of graphic charts
- § opening of pop-up windows at the server's discretion
- § automatic management of meta-push events
- § callbacks for custom management of events.

Desktop applications

Lightstreamer is also the ideal solution to feed **client desktop applications** in streaming/push mode, in other words the classic window-based applications that can be written in various languages (**Java**, **Visual Basic**, **C/C++**, **C#**, etc.) and which require an installation on users' PCs, or applets and ActiveX controls that don't require installation. Lightstreamer's powerful bandwidth & frequency management, adaptive streaming and multiplexing mechanisms are indispensable for any kind of application that needs to receive a flow of asynchronous data.

An excellent example of how **Lightstreamer Server** can be used to feed a sophisticated application clients is the **T3** system introduced by **IntesaTrade**[5]. T3 is an online trading desktop written in Java that receives real-time market, order and portfolio data from Lightstreamer Server.



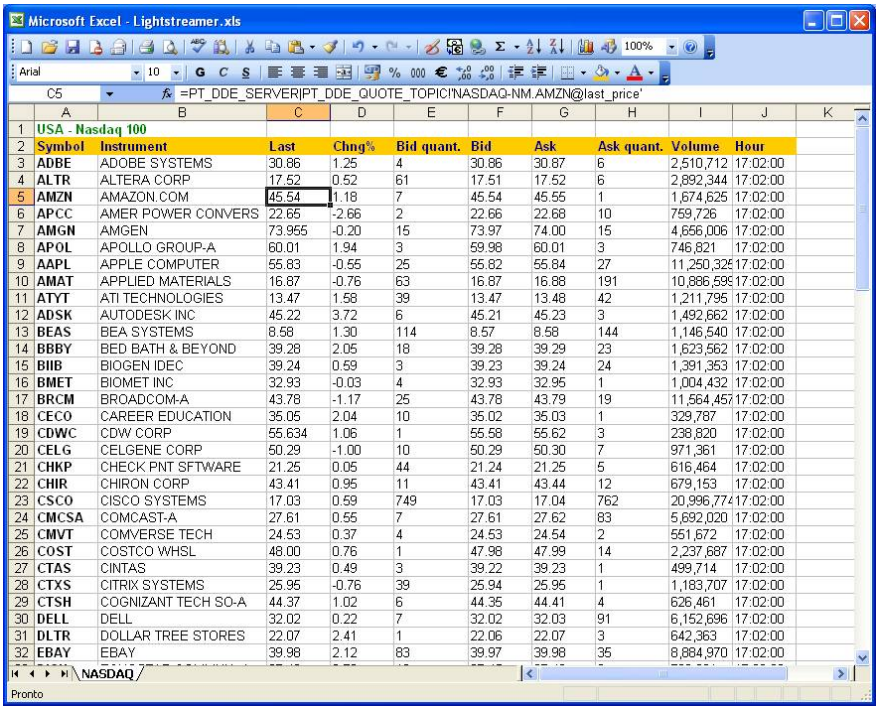
Of course, it is also possible to write Lightstreamer application clients in the form of **Java applets** or **Flash animations** to be slotted into web pages in

[5] www.intesatrade.it (a company that is part of the Italian group Banca Intesa).

all cases where, for whatever reason, it is decided not to build an entirely HTML front-end.

Microsoft Excel

Lightstreamer also makes it possible to push feed an **Excel spreadsheet** created by the user. A **DDE** (*Dynamic Data Exchange*) gateway receives the data in streaming mode from Lightstreamer Server and makes them available to the Excel application (or to other DDE clients). In this way, the figures contained in certain cells of the spreadsheet change dynamically in real time, in exactly the same way as for other kinds of Lightstreamer clients.



	A	B	C	D	E	F	G	H	I	J
1	USA - Nasdaq 100									
2	Symbol	Instrument	Last	Chng%	Bid quant.	Bid	Ask	Ask quant.	Volume	Hour
3	ADBE	ADOBE SYSTEMS	30.86	1.25	4	30.86	30.87	6	2,510,712	17:02:00
4	ALTR	ALTERA CORP	17.52	0.52	61	17.51	17.52	6	2,892,344	17:02:00
5	AMZN	AMAZON.COM	45.54	1.18	7	45.54	45.55	1	1,674,625	17:02:00
6	APCC	AMER POWER CONVERS	22.65	-2.66	2	22.66	22.68	10	759,726	17:02:00
7	AMGN	AMGEN	73.955	-0.20	15	73.97	74.00	15	4,656,006	17:02:00
8	APOL	APOLLO GROUP-A	60.01	1.94	3	59.98	60.01	3	746,821	17:02:00
9	AAPL	APPLE COMPUTER	55.83	-0.55	25	55.82	55.84	27	11,250,325	17:02:00
10	AMAT	APPLIED MATERIALS	16.87	-0.76	63	16.87	16.88	191	10,886,595	17:02:00
11	ATYT	ATI TECHNOLOGIES	13.47	1.58	39	13.47	13.48	42	1,211,795	17:02:00
12	ADSK	AUTODESK INC	45.22	3.72	6	45.21	45.23	3	1,492,662	17:02:00
13	BEAS	BEA SYSTEMS	8.58	1.30	114	8.57	8.58	144	1,146,540	17:02:00
14	BBBY	BED BATH & BEYOND	39.28	2.05	18	39.28	39.29	23	1,623,562	17:02:00
15	BIIB	BIOGEN IDEC	39.24	0.59	3	39.23	39.24	24	1,391,353	17:02:00
16	BMET	BIOMET INC	32.93	-0.03	4	32.93	32.95	1	1,004,432	17:02:00
17	BRCM	BROADCOM-A	43.78	-1.17	25	43.78	43.79	19	11,564,457	17:02:00
18	CECO	CAREER EDUCATION	35.05	2.04	10	35.02	35.03	1	329,787	17:02:00
19	CDWC	CDW CORP	55.634	1.06	1	55.58	55.62	3	238,820	17:02:00
20	CELG	CELGENE CORP	50.29	-1.00	10	50.29	50.30	7	971,361	17:02:00
21	CHKP	CHECK PNT SFTWARE	21.25	0.05	44	21.24	21.25	5	616,464	17:02:00
22	CHIR	CHIRON CORP	43.41	0.95	11	43.41	43.44	12	679,153	17:02:00
23	CSCO	CISCO SYSTEMS	17.03	0.59	749	17.03	17.04	762	20,996,774	17:02:00
24	CMCSA	COMCAST-A	27.61	0.55	7	27.61	27.62	83	5,692,020	17:02:00
25	CMVT	COMVERSE TECH	24.53	0.37	4	24.53	24.54	2	551,672	17:02:00
26	COST	COSTCO WHSL	48.00	0.76	1	47.98	47.99	14	2,237,687	17:02:00
27	CTAS	CINTAS	39.23	0.49	3	39.22	39.23	1	499,714	17:02:00
28	CTXS	CITRIX SYSTEMS	25.95	-0.76	39	25.94	25.95	1	1,183,707	17:02:00
29	CTSH	COGNIZANT TECH SO-A	44.37	1.02	6	44.35	44.41	4	626,461	17:02:00
30	DELL	DELL	32.02	0.22	7	32.02	32.03	91	6,152,696	17:02:00
31	DLTR	DOLLAR TREE STORES	22.07	2.41	1	22.06	22.07	3	642,363	17:02:00
32	EBAY	EBAY	39.98	2.12	83	39.97	39.98	35	8,884,970	17:02:00

Java Midlets

The **J2ME** (*Java 2 Micro Edition*) technology has for some time been available on all of the main models of mobile phone, offering the chance to distribute **midlets written in Java** for execution directly on mobiles. The Java midlet market has turned out to be extremely interesting and full of initiatives.

Setting up Lightstreamer clients in the form of midlets is immediate. This makes it possible to have a mobile application that presents updates in streaming/push, taking advantage of Lightstreamer's powerful bandwidth control mechanisms, which are seen to be even more vital in a mobile environment.

Opera browsers for smartphones & PDAs

Opera Software[6] provides a version of its famous web browser specifically downsized for mobile devices (smartphones and PDAs). By means of its *SSR* (Small Screen Rendering) technology, the Opera browser is able to visualize normal HTML pages on the small displays of last-generation mobile phones.
In its continuous evolution to keep up with the diffusion of any new web access channel, Lightstreamer also supports pure streaming/push in HTML on Opera Mobile browsers. This is an important innovation, as it extends the Lightstreamer model to portable devices as well: i.e. the ability to visualize data in real time with the same ease and immediacy as visiting a normal web page. There is no need to install any extra software on the mobile phone to use the push service (as Opera is pre-installed by the principal mobile phone manufacturers).

Heterogeneous channels

The various combinations of devices and software that receive push data can connect to Lightstreamer Server through heterogeneous channels. The only prerequisite is that there should be a TCP/IP network.
Lightstreamer is therefore suitable for serving various kinds of channels:

- **Intranet**: Lightstreamer can be used to distribute live data within a corporate LAN or private geographical network (e.g. providing real-time financial information to the branches of a bank).
- **Internet**: Lightstreamer is optimized for the distribution of real-time data on standard Internet connections, which can take place through any mode of access (analog modem, ADSL modem, optical fiber, dedicated line, etc.).
- **Mobile Internet**: Lightstreamer's light weight, bandwidth control and automatic adaptation to dynamic variations in the behavior of the network make it indispensable for streaming/push on wireless Internet connections (based on GSM, GPRS, UMTS, WI-FI, etc.).

Types of applications

The Lightstreamer technology can be used in many applications. It is the right solution in all those cases where there is a need to send users

[6] www.opera.com

updated information in real time through a web browser, in the simplest and most efficient way possible.
The following are some examples of application domains that are likely candidates for drawing the maximum benefits from integration with Lightstreamer.

Financial information

The primary targets for these kinds of applications are **banks**, **security houses**, **information providers** and **portals** that want to provide users with real-time financial information or **online trading** services. The client can be application-based or web-based. The types of information that are suitable for being provided in push include:

- **Quotes** provided in real time for shares, covered warrants, derivatives, fx, etc.
- **Multi-level book** for financial instruments.
- **News headlines** published by press agencies.
- **Indices** and **exchange rates** valued in real time.
- Dynamic changes in the composition of **share lists** (e.g. top/bottom performers of the day).
- **Order monitor** with real-time updates on the state of orders placed (submitted, executed, cancelled, etc.).
- **User portfolio** with automatic opening, closure and update of positions.

News

With Lightstreamer it is possible to create web pages with one or more lists of **news headlines** updated in push. The user can see a news item on a dynamic basis as soon as it is published by the press agency, without having to refresh the page (manually or automatically, as happens with most sites).
Lightstreamer therefore eliminates any delay between the generation of a headline (especially of the **"breaking news"** variety) and its visualization on a web page that is already open. It is like having a direct link with the press agencies through a normal browser and an Internet connection.
It is possible to connect Lightstreamer to any kind of feed, simply by writing specific adapters. For example, Lightstreamer can be integrated with the Internet open feeds based on **RSS** (*Really Simple Syndication*) and **Atom** standards.

Monitoring consoles

There are numerous hardware and software platforms on the market with webpage-based **administration consoles**. Generally speaking, these are **system / network / application / security / performance management** systems that could be improved through Lightstreamer.
For example, integrating Lightstreamer into an **application server** equipped with a web-based monitoring system makes it possible to keep critical system information (such as the number of connections, amount of memory available, size of the pool of resources, etc.) under control without having to refresh the page and without having to introduce applets.
Another example is the introduction of Lightstreamer into the web console of a **firewall,** which makes it possible to control all of the vital security

parameters in real time simply by loading a web page onto any kind of browser.
Lightstreamer can be integrated with any type of system that requires **remote monitoring** (including, for example, **industrial machines**). Push updates of information are also possible on **mobile devices**, for those cases where there is an additional requirement to monitor the system while the user is on the move.

Online betting

Online betting portals are highly effective if **real-time odds** can be displayed. With Lightstreamer any kind of betting (sports, financial, politics, etc.) can be enlivened with real-time streaming data.

Online games, gambling

Portals that run online games can also benefit from Lightstreamer's push engine. **Bingo**, **roulette** and other games can be offered online without having to download any software, while maintaining full and genuine **interactivity** between the players that are online.

Online auctions

Online auctions have had considerable success. With Lightstreamer it is possible to organize **auctions that only last a few minutes**, just like traditional auctions, thanks to the real interactivity that participants would have. Each bid made by a participant gets communicated instantly to all of the others through a simple HTML page. Lightstreamer can be applied both to **"English" auctions**, where the price goes up, and to **"Dutch" auctions**, where the price goes down (like those organized by certain government entities).

Sports portals

Lightstreamer makes it possible to update **sporting results** live on any kind of browser. For example, as regards **football**, on any one day of the league championship it is possible to visualize an HTML page with a table that shows all of the goals that have been scored as well as other statistical data of the games being played. Any time the data has to be updated, the information is sent automatically to all of the browsers that are connected. As always, the Lightstreamer pages are *live* without having to download any external element, while they can be used by PCs as well as by smartphones.
Of course, there are many other sports that lend themselves to applications of this kind, because there is a continuous flow of information that is generated in real time during the match. Another example is **Formula 1**. With Lightstreamer it is possible to send **telemetry data** to users who are connected to a normal website.

Chat Systems, Instant Messengers, Online Communities, Social Networks

Current **chat** systems generally require a **Java applet**, an **ActiveX** control or a **Flash** component to be loaded onto the user's browser. This tends to be an obstacle that gets in the way of wider diffusion. Indeed, problems

often occur in connection with the required Java virtual machine or with the digital signature and certification of the downloaded code. With Lightstreamer it is possible to create a complete and functionally rich web chat system based entirely on standard HTML pages. When the user accesses the chat section, the application gets automatically launched in the same way that any other page on the site gets visualized, without any need for special action (such as clicking on pop-up windows which need permission to download or a security alert).
The same considerations apply to **instant messaging** systems. Lightstreamer makes it possible to connect immediately to a network that manages online messages through an HTML page, which can be loaded either onto a PC browser or onto a mobile device browser, without having to install any kind of software.
Lastly, just think of the numerous sites that offer the chance to participate in **online communities** and **social networks**. Having accessed the site by inputting a nickname and password, users can see which other users of that community are online (presence), review their profile, interact symbolically or send messages. **An online community is all the more powerful the higher the level of interactivity that is offered.** To find out if new online users have joined the site or if other users have interacted with us, the page usually has to be refreshed (automatically or manually). But with Lightstreamer it is possible to send such information **in real time and without any delays** to all connected users.

Website access statistics

Many systems and services for website access statistics offer web pages with **real-time reporting of the number of visits made to a website**. All of these statistics (number of visitors, most requested pages, top referring sites, list of clients, top paths through site, etc.) are currently offered in pull mode. With Lightstreamer they would be updated dynamically on the HTML page, without having to reload periodically.

Web-Mail systems

Many people use e-mail clients based on HTML pages, instead of an application client installed on their PC. Such **Web Mail** systems periodically need to reload the page in order to visualize any new e-mails that have arrived. With Lightstreamer it is possible to load the page once onto the browser and keep the window open for the entire day. Each new e-mail then gets visualized dynamically in push as soon as it arrives.

Transport websites

Another area where dynamic and automatic updates are fundamental is that of **transport timetables**. All companies and organizations that operate in the transport field (**airports**, **railway stations**, **seaports**, etc.) publish their timetables on the web. With Lightstreamer, timetables that are visualized in HTML can be updated in real time, indicating late arrivals and any changes in the timetable. As a result it is possible to make a web page very similar to the classic arrivals and departures board found at airports and railway stations.
Lightstreamer can also be used to update **road and motorway traffic information**, alerting users of any emergencies.

Public utility services

There are numerous **government entities**, sectors of the **public administration** and **non-profit organizations** that use Internet to publish information that gets constantly updated. For example, certain **emergency services** publish real-time statistics on their current operations (position of ambulances, types of emergencies being dealt with, etc.). Lightstreamer can considerably improve the communication of such information, thereby creating benefits in the **healthcare sector** (**local health authorities, hospitals, ambulances, first-aid services, clinics, research centers, etc**.).
In addition, just think of all those cases where the rapid evolution of natural phenomena requires real-time communication of news bulletins and other information by the **Civil Protection Authority**. The same applies to real-time communications by the **Fire Brigade**, **Forestry Guards**, **Police**, **Army**, **Navy**, etc.

Availability of materials – ticket offices – tourism

With Lightstreamer it is possible to show in real time the effective availability of any kind of material, resource or object.
For example, in an **industrial environment** it would be possible to check **inventory levels** through a browser.
Alternatively, in the **entertainment sector** it would be possible to consult the availability of **tickets for the cinema, theater, concerts or sports events ("online ticketing")**.
And in the **tourist industry** it would be possible to consult in real time the availability of **hotel rooms, air tickets, train tickets**, etc.

Opinion polls

Live web as permitted by Lightstreamer is an excellent way to publish the results of **opinion polls**, whether online polls, or polls carried out through more traditional channels (telephone calls, interviews, etc). For example, in an **election environment**, it would be possible to communicate all updates of **exit polls** and **ballot counts** in real time via browser.

Other applications

There is an almost infinite number of applications for this push technology based on web pages. Suffice to think of **sales results**, **real-time reporting for business intelligence systems**, **weather reports**, **e-learning and remote training**, **probe output, online tracking**, etc.

User "permanence" on a website

One thing that is thoroughly innovative is the ability to make a user's *permanence* on a website real. With normal HTML pages that are not connected to Lightstreamer the user is "present" on the site only when they load a new page. Once this phase is over, all points of contact between the browser and the remote site disappear, until such time that another page is loaded or the same page is refreshed. In other words, under the usual

model of web surfing the concept of "**physical presence**" on a site does not exist, only "**logical presence**" (i.e. a user is deemed to be present on a site for a set period of time calculated from the moment that they load the last page).
Lightstreamer makes it possible to change this surfing paradigm quite radically. The user loads the home page of a website onto their browser. This page (as always without any applets, plug-ins or elements other than HTML/JavaScript) activates a permanent connection with Lightstreamer in a frame. The user is therefore free to surf anywhere they want in the site without the connection with Lightstreamer ever being interrupted. This means having a physical channel permanently available for the flow of information that is totally independent from the page of the site that the user is reading at the time. This makes it possible for example to show **alerts** on the browser at any moment in time, at the server's discretion. This means that the user is ***reachable*** for the entire period of their permanence on the website.
Consider the case of the administrator of a *system* (of any kind) that connects with a web console that reports summary data on the state of the system. If **alarms** have to be sent out when certain conditions exist, without Lightstreamer recourse would have to be had to e-mail, SMS or similar methods of communication. With Lightstreamer, on the other hand, at the very moment that the condition exists for an alarm to be sent out, it is possible to make a **pop-up window** appear on the user's browser, without having to wait for a page to be reloaded and without having to download applets.
It is possible to imagine numerous situations in which this technology could be used for **marketing** purposes as well.

4. What Lightstreamer is offering

Lightstreamer is a modular solution that can satisfy the needs of large, medium and small businesses. Three editions are available: **Entry**, **Standard** and **Enterprise**. The product suite encompasses the Server Engines, the Client Engines and several add-ons components conceived to simplify both back-end and front-end integration:

- **Lightstreamer Server**: the push server engine, around which a personalized streaming/push solution can be built.
- **SDK for Data Adapter and Metadata Adapter Development**: the resources (libraries, documentation, example) needed to create customized Data Adapters and Metadata Adapters to integrate Lightstreamer Server with the data feeds.
- **SDK for Web Client Development**: existing Web front-ends can become "Lightstreamer-enabled" by integrating the Lightstreamer Web Client JavaScript libraries (provided as part of this SDK) with the existing pages.
- **SDK for Java J2SE Client Development**: in the case where the client is not a Web browser but a Java application, a Java Client Library is provided as part of this SDK. This library completely hides the network protocol of Lightstreamer and offers a higher level of abstraction to the developers that need to incorporate streaming data in their Java applications.
- **Lightstreamer DDE Gateway**: a proxy module that implements a DDE Server that broadcast the real-time events received by Lightstreamer Server. The data can be delivered to any DDE client, such as Microsoft Excel.
- **Lightstreamer Remote Adapter**: a module that allows Lightstreamer Adapters to be run outside Lightstreamer Server process. Through the Remote Adapter, custom adapters can reside on remote machines (e.g. behind the DMZ).
- **Lightstreamer COM Adapter**: an Adapter that expses Lightstreamer Adapter interfaces as COM interfaces.

How to integrate Lightstreamer into an IT system

Integrating Lightstreamer into a new or existing system is generally quite straightforward, inexpensive and low risk. After you have bought Lightstreamer Server (the essential module), the steps to be taken are the following:

- **Integration with the data feeds**: this entails developing one or more Data and Metadata Adapters to hook up Lightstreamer Server to the specific data feeds in the case in point. The Adapters have to be developed using a **Java** API that is provided along with Lightstreamer Server. The code used in the Adapters is then free to employ any kind of middleware for integration with the data feeds (JMS, JDBC, CORBA, RendezVous, sockets, etc.) or to call on native codes or libraries (through JNI).

- **Integration with the front-end**: in the event that the user decides to go the route of an *HTML front-end* for Lightstreamer, the process of integration envisages the use of **JavaScript libraries** that are provided along with Lightstreamer Server, through which it is possible to make the web pages "Lightstreamer enabled". On the other hand, if the user wants to create an *application client*, integration merely consists of adjusting the client to the network protocol of Lightstreamer (this can be done using a **Java client library** that is supplied along with Lightstreamer or directly implementing the simple network protocol of Lightstreamer).

Contacts

web site:
www.lightstreamer.com

email:
info@lightstreamer.com

CTO

Alessandro Alinone
alessandro.alinone@lightstreamer.com
Tel: +39 02 66732 1

VP Sales

Simon Walmsley
simon.walmsley@lightstreamer.com
Tel: +39 02 66732 1