Lightstreamer

The Streaming-Ajax Revolution

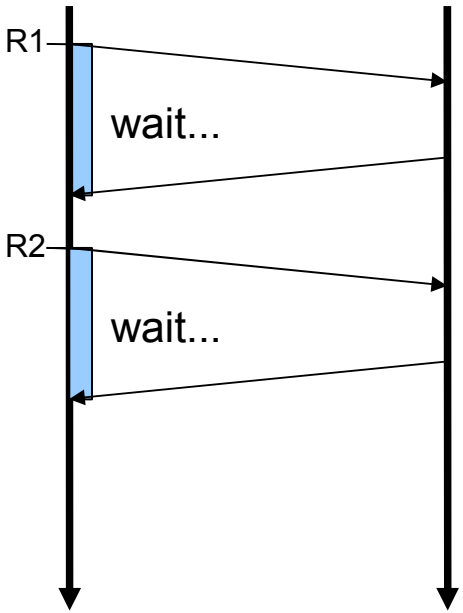
Product Insight

Agenda

- Paradigms for the Real-Time Web (four models explained)
- Requirements for a Good Comet Solution
- Introduction to Lightstreamer
- Lightstreamer Demonstrations
- Lightstreamer Architecture
- Connection Management
- Bandwidth Management
- Integration Steps

Intro: Synchronous & Asynchronous Modes

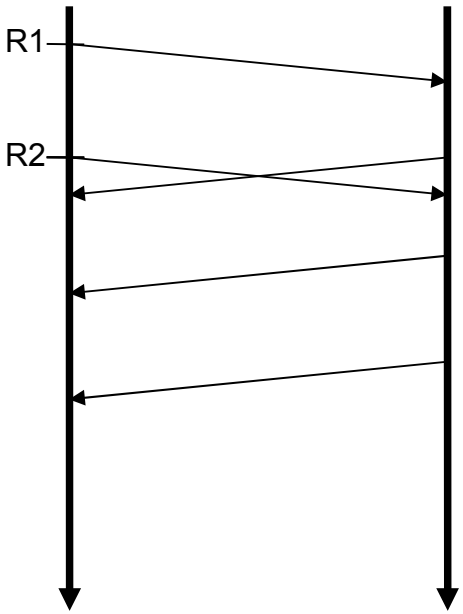
Synchronous mode
of a server (S)
with respect to an actor (A)



actions by A
through time
(**requests**)

actions by S
through time
(**responses**)

Asynchronous mode
of a server (S)
with respect to an actor (A)



actions by A
through time
(**requests**)

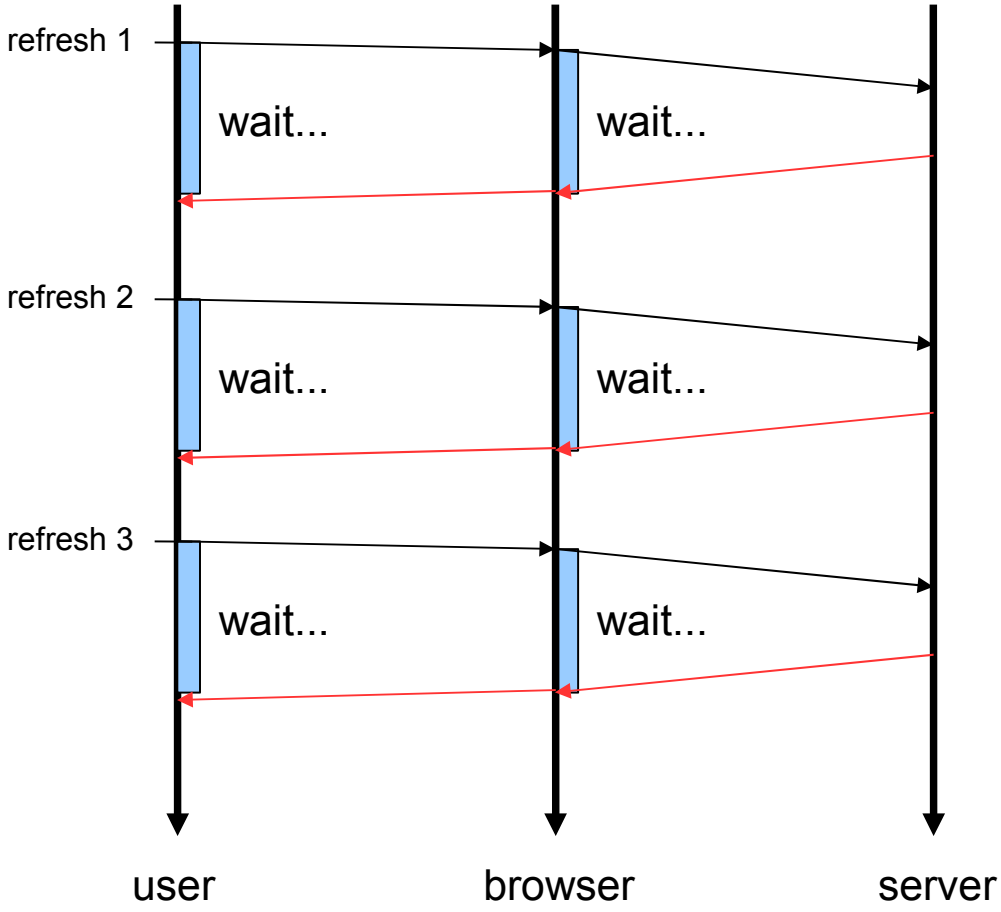
actions by S
through time
(**responses**)

Paradigms for the Real-Time Web

	Application Paradigm	Method of sending data with respect to the user's actions	Method of sending data with respect to the browser's actions
1	Traditional Web Application → Page Refresh	Synchronous	Synchronous
2	Classic Ajax Application → Periodic Polling	Asynchronous	Synchronous
3	Smart Ajax Application (Lightstreamer Application) → Smart Polling → Asynchronous Polling → Comet - Long Poll → Ajax Push	Asynchronous	Partially Asynchronous
4	Streaming Ajax Application (Lightstreamer Application) → Streaming Ajax → True Push/Streaming → Comet - Forever Frame → Reverse Ajax (term used for polling too)	Asynchronous	Asynchronous

Model 1: Web Application Based on Refresh

Manual or automatic refresh

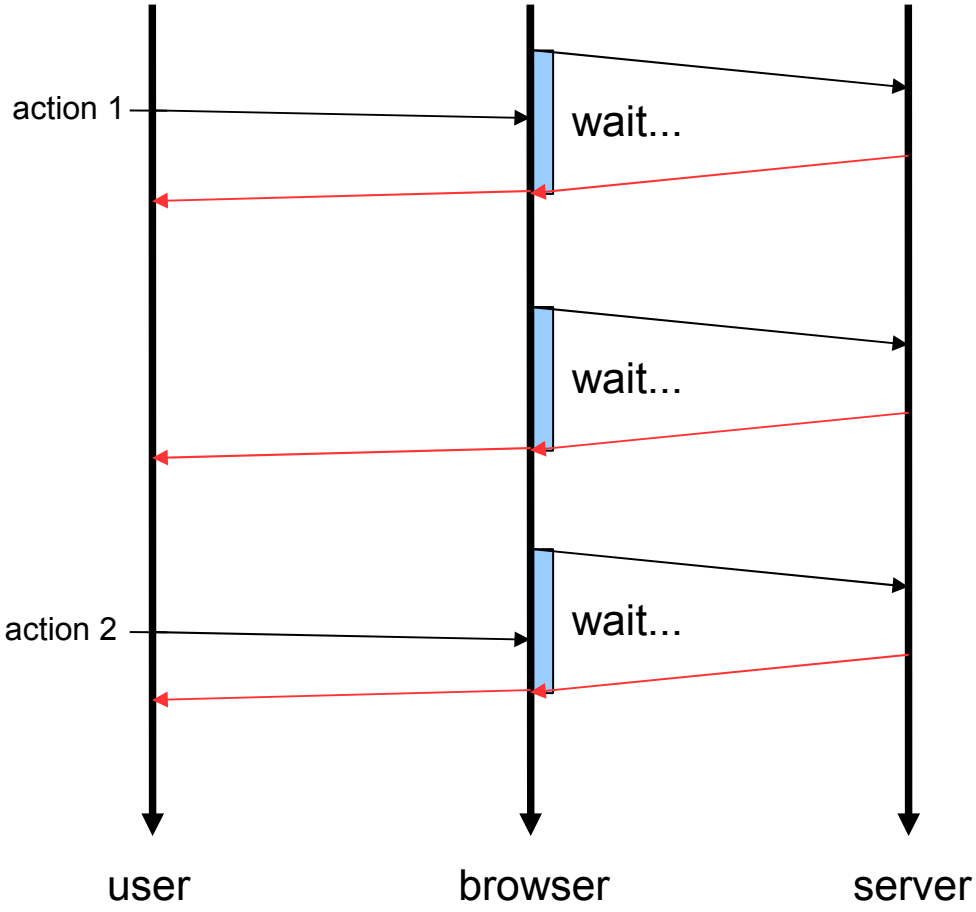


Typical issues:

- ✔ Low update frequency; no real time
- ✔ High bandwidth usage
- ✔ High load on Web server

Model 2: Ajax Application Based on Polling

Periodic polling done by the Ajax layer



Typical issues:

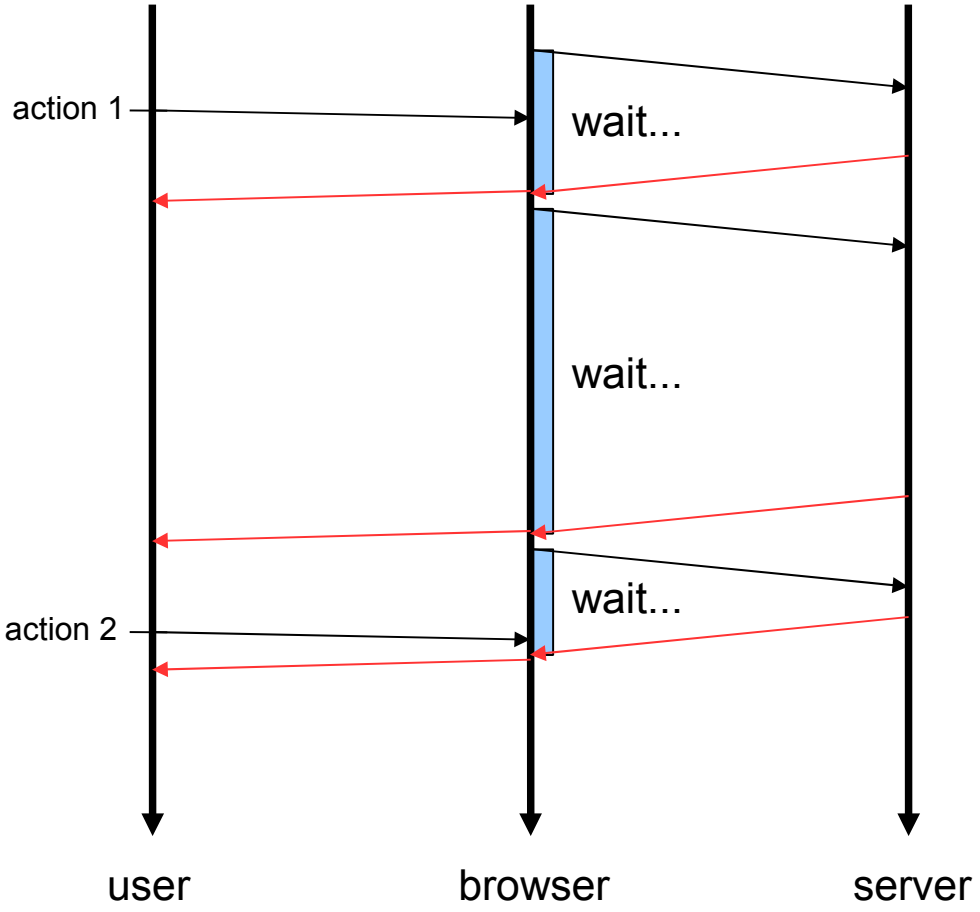
- ✔ Low update frequency; no real time
- ✔ High bandwidth usage (but potentially lower than model 1)
- ✔ High load on Web server

Advantages:

- ✔ User interface never blocked

Model 3: Ajax-Comet Application Based on Smart (Long) Polling

Asynchronous polling (variable polling frequency, controlled by server)



Typical issues:

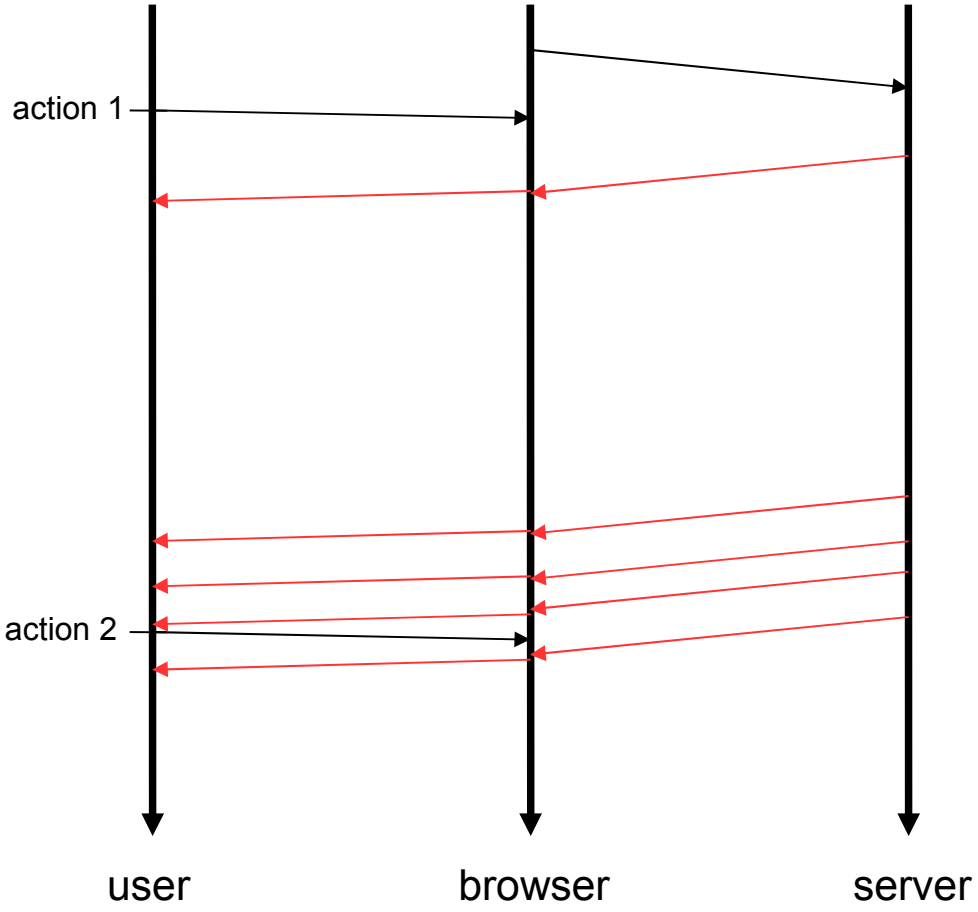
- ✔ Medium update frequency; near real time
- ✔ Medium bandwidth usage (HTTP headers present in each round-trip cycle)
- ✔ High load on Web server

Advantages:

- ✔ User interface never blocked
- ✔ Zero latency on low-frequency events

Model 4: Ajax-Comet Application Based on Streaming

Streaming Ajax (fully asynchronous push)



Typical issues:

- ✔ Blocked by some anti-virus software mounted on proxy servers

Advantages:

- ✔ High update frequency; true real time
- ✔ Low bandwidth usage (very little overhead)
- ✔ Low load on the network infrastructure

Requirements for a Good Comet Solution

- **Scalability:** should scale well to thousands of concurrent push sessions (the architecture of traditional web/application servers is not suitable for models 3 and 4).
- **Universality:** should pass through the majority of firewalls and proxy servers and should support the broadest range of web browsers and AJAX toolkits & frameworks.
- **Network Lightness:** should employ a network protocol that minimizes the used bandwidth.
- **Network Adaptiveness:** should dynamically adapt to the changes in network conditions.
- **Quality of Service:** should allow to allocate a maximum bandwidth and a maximum update frequency for each push channel.
- **Data Flexibility:** should support multiple push & filtering modes based on the intrinsic nature of the data.
- **Robustness:** should support transparent fail-over.

What is Lightstreamer?

- Lightstreamer is a **push/streaming** engine designed for **low latency** and **high scalability**.
- Conceived in year 2000, has more than **eight years of production maturity** on the market (especially within mission-critical systems in the financial industry).
- Implements a **pub/sub middleware** based on HTTP and supports **models 3 and 4**.
- Fully manages **bandwidth**, **frequency** and **network congestions**.
- Supports **many types of clients**, including zero-install **AJAX and Flex applications**.
- Provides **full SDKs** on both server-side and client-side to integrate with any kind of system.

Lightstreamer Demos

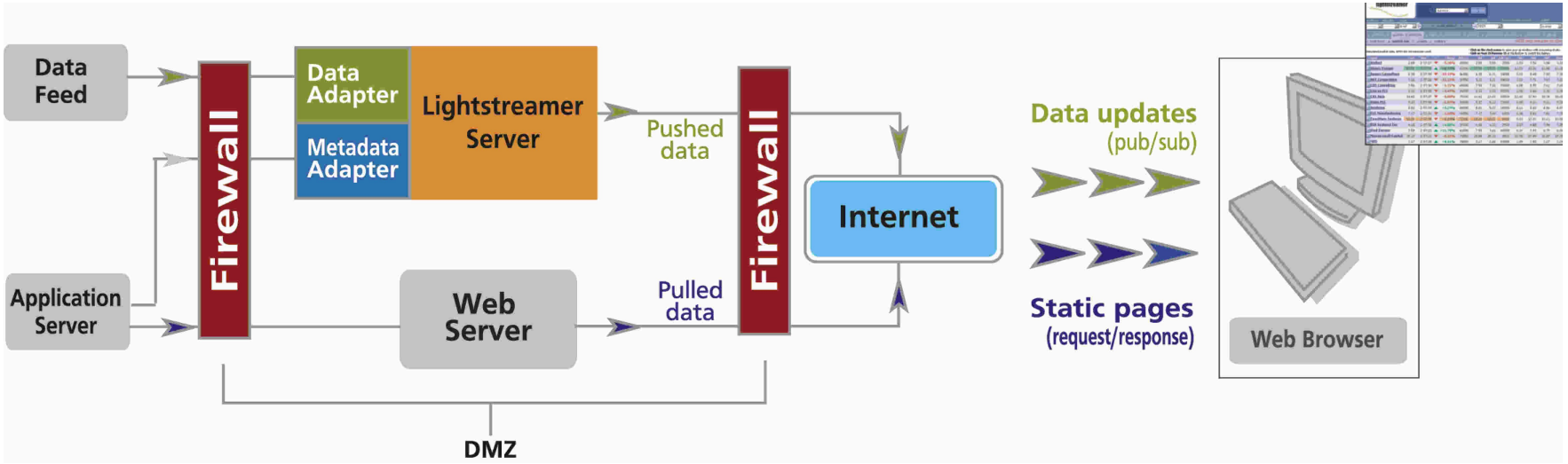
>> www.lightstreamer.com/demo <<

Lightstreamer's Architecture

- Lightstreamer Server is a stand-alone Java process.
- Back-end integration through custom Adapters written with Java, .NET, or direct TCP sockets.

Front-end integration with any kind of client technology:

- **Thin Client**: Through *JavaScript* API compatible with any web browsers and integrable with any AJAX frameworks. Through *ActionScript* API for Flex applications.
- **Thick Client**: through *Java* and *.NET* API or through HTTP-based network protocol.



How Lightstreamer Handles Connections

- **Stream Connection**
  - Single permanent **Http/Https** connection through which JavaScript commands are delivered to the browser in real-time (*model 4*). It is kept open even through proxy servers and firewalls.
  - **Multiplexing** techniques are used to deliver the data destined to multiple frames and windows over a single physical connection, in order to avoid saturating the connection pool of the browser.
- **Control Connections**
  - Short-lived connections mainly used to deliver subscribe and unsubscribe commands to the Server.
- **Stream-Sense**
  - Http streaming usually passes through most proxies and firewalls But there exist cases where streaming is blocked by a particular combination of proxy and antivirus software.
  - The “Stream-Sense” feature detects when streaming is blocked and automatically switches to **smart polling** (*model 3*).

How Lightstreamer Scales

- **Staged Event-Driven Architecture**
  - **Thread pools** of fixed size to handle an arbitrary number of concurrent connections. Complete decoupling between threads and sockets.
  - **Non-blocking I/O**, based on **Java NIO**, used for all types of connections.
  - **Graceful degradation** of the quality of service in the case that the Server's CPU is saturated.
- **Scalability**
  - **Vertical scalability:** an instance of Lightstreamer Server can fully leverage multiple CPUs and cores available in a box.
  - The number of **concurrent sessions** handled by a Server instance depends on several variables, in particular: inbound throughput, outbound throughput, payload, subscription mode.
  - **Horizontal scalability:** a cluster of Lightstreamer Servers can be easily implemented through a standard Load Balancer.

How Lightstreamer Manages Bandwidth (1)

- **Data Filterability**
  - The nature of some data enables filterability.
  - Lightstreamer's data filtering always ensures consistency and completeness.
  - Different subscription modes are available, based on the data nature (Merge, Distinct, Raw, Command --> ***metapush***).
- **Bandwidth Control**
  - A maximum bandwidth can be allocated for each user. Data is dispatched based on the configured bandwidth.
  - Bandwidth control is available in both streaming mode and smart-polling mode.
- **Frequency Control**
  - Each single subscription can request a maximum update frequency.
  - Frequency control is available in both streaming mode and smart-polling mode.

How Lightstreamer Manages Bandwidth (2)

- **Multi-Stage Filtering**
  - A Pre-Filter is available to preliminarily sample the data on a global basis.
- **Adaptive Streaming**
  - Lightstreamer automatically detects Internet congestions and heuristically throttles the data flow based on the available bandwidth.
  - When the network channel is fully available again, the user will not receive a burst of old updates but will start seeing fresh data at once (i.e. data aging is avoided).
  - Data is aggregated efficiently within TCP packets, with a direct control over the number of sent packets (a trade-off between latency and overhead reduction can be configured).
  - Adaptive streaming is particularly useful for streaming sessions held over **mobile networks** or any unreliable networks.

How to Integrate Lightstreamer in a System

- **Server-Side Development**
  - Develop a **Data Adapter** to connect Lightstreamer Server to your data source.
  - Develop a **Metadata Adapter**, if you need to manage authentication and authorization.
  - The Adapters can be developed in **Java** or **.NET** through the provided APIs, or in any other language at **TCP-socket** level.
- **Client-Side Development**
  - **Web Client:** Include the provided JavaScript libraries in the pages of the Web application.
    - The JS libraries can coexist with third-party JavaScript or AJAX frameworks and toolkits (e.g. Lightstreamer + TIBCO General Interface; Lightstreamer + Dojo; Lightstreamer + ASP.NET AJAX).
  - **Flex Client:** Include the provided ActionScript library in your application.
  - **Thick Clients:** Use a provided client library within the client application (Java SE, Java ME, .NET, iPhone) or implement the protocol at HTTP level in any language.

lightstreamer

www.lightstreamer.com

info@lightstreamer.com