From Data Push to WebSockets

The History of Data Push and the Lightstreamer Technology

Alessandro Alinone

Last updated: 17 Oct 2018

Agenda

- History of Data Push
- Lightstreamer Technology
- Lightstreamer Success Stories
- MQTT.Cool and JMS Extender

What Is Data Push?

Information is delivered on the fly as soon as it is generated. Web pages and mobile apps are updated in real time.

Many application domains are taking benefit from push technology:

- Financial Services: Online trading platforms for capital markets, live price dissemination, order submission, spread betting
- Gaming: Sports betting, online casinos, online multiplayer video games
- Aerospace and Defense: Web telemetry of space vehicles, satellites, and aircrafts, web-based management of airport operations
- Media: social TV, second screen, sports event live data
- Transportation and Logistics: live tracking, supply chain monitoring
- Alerting: Emergency mass notification systems
- And many others: Social networks, in-app notifications, online collaboration tools, online auctions, systems monitoring, e-learning, etc.

Many Terms Used to Refer to Data Push

Real-Time Messaging

Data Streaming

In-App Messaging

Push Technology

WebSockets

In-App Notifications

Comet

Real-Time Web

Push Notifications

Real-Time Notifications

Long Polling

Web Streaming

Data Push

Internet Messaging

Last Mile Messaging

Reverse Ajax

Ajax Push

Web Push

and others...

The Four Waves of Data Push

- **1996-2000**: Webcasting
  Coarse-grained daily updates

- **2000-2012**: Comet
  (the term "Comet" was coined by Alex Russell in 2006, but Comet technologies existed since 2000)
  Polling, long polling, streaming

- **2009 onwards**: Push Notifications
  Apple APNs, Google FCM (previously C2DM and GCM), Web Push protocol

- **2012 onwards**: WebSockets
  Full-duplex bidirectional streaming

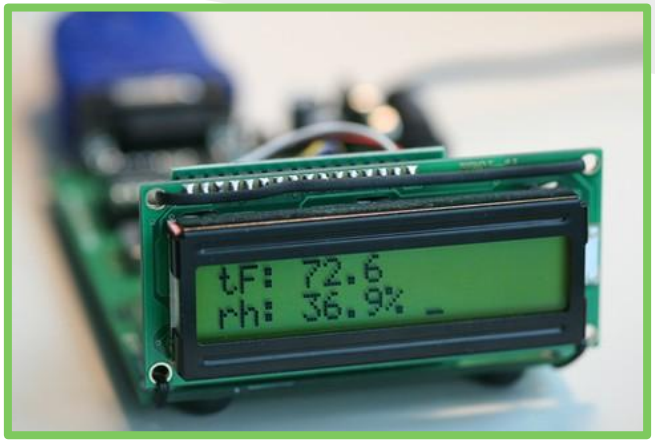
Second Wave: the Raise of Data Push

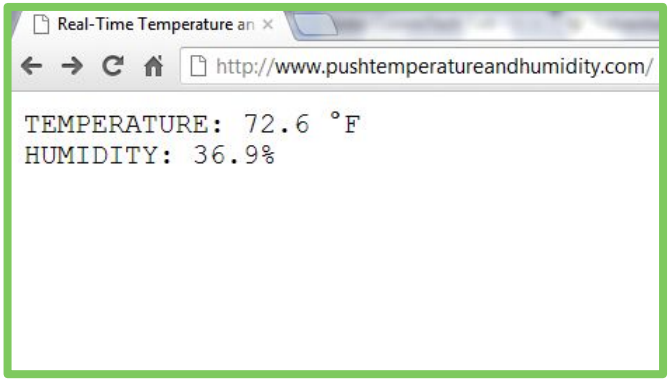
- **2000**: Online financial trading systems required data push for real-time stock price delivery
- Requirements:
  - Fine-grained updates
  - Real-time updates (low latency)
- Very first players: Lightstreamer, Caplin, Pushlets, KnowNow
- Technology:
  - Front-end: HTML and/or Java applets
  - Transport techniques: **Ajax polling**, **Comet long polling**, and **Comet streaming**

An Example to Help Illustrate

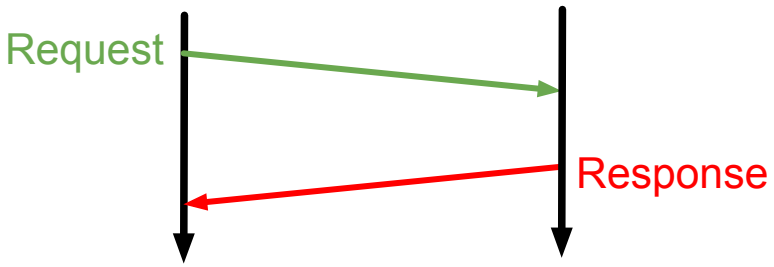
A temperature and humidity sensor must send data to a Web browser (***sensor example***).

Let's see how this might have been done in the history of push technology.





HTTP/1.1



Request

GET / HTTP/1.1
Host: www.facebook.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:16.0) Gecko/20100101 Firefox/16.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: datr=IeCPUJWOBWaU0LrmpOTOC-YX; reg_fb_gate=http%3A%2F%2Fwww.facebook.com%2F; reg_fb_ref=http%3A%2F%2Fwww.facebook.com%2F; wd=1080x1281
Cache-Control: max-age=0

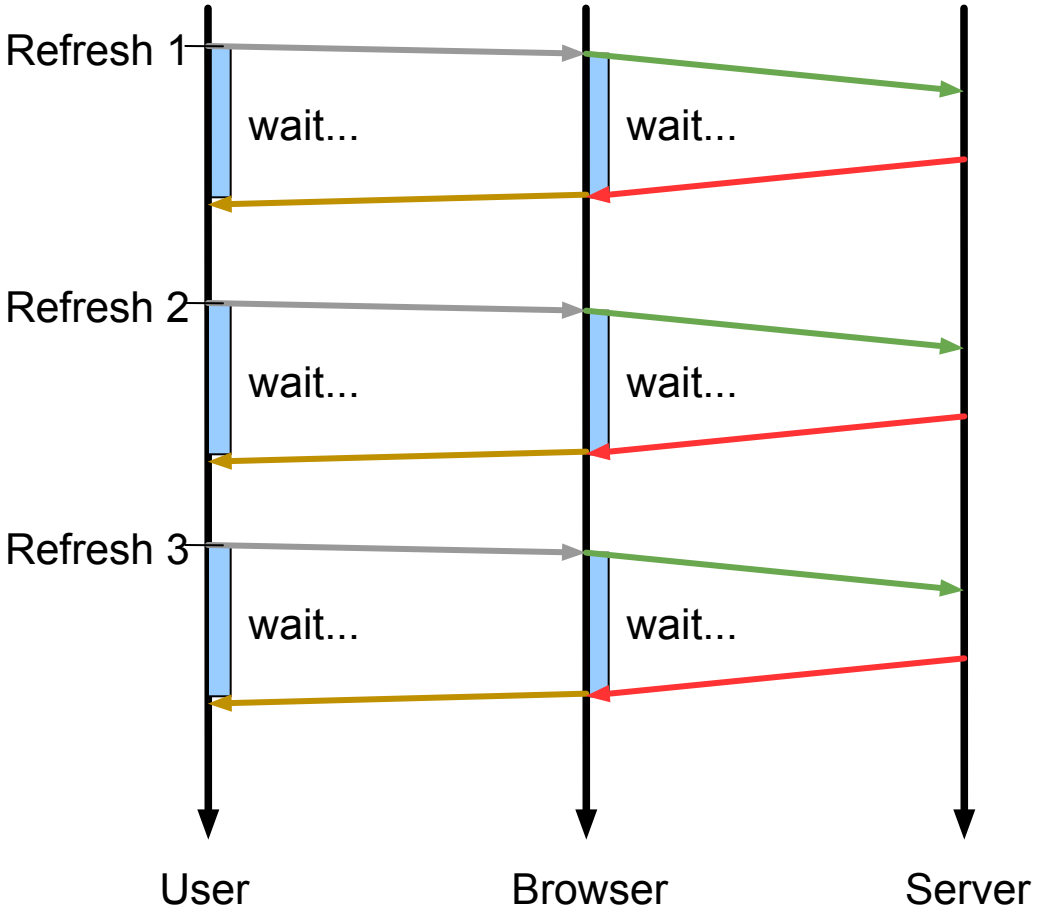
Response

HTTP/1.1 200 OK
Cache-Control: private, no-cache, no-store, must-revalidate
Expires: Sat, 01 Jan 2000 00:00:00 GMT
P3P: CP="Facebook does not have a P3P policy. Learn why here: http://fb.me/p3p"
Pragma: no-cache
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Set-Cookie: reg_ext_ref=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; path=/; domain=.facebook.com
Set-Cookie: wd=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; path=/; domain=.facebook.com; httponly
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
X-FB-Debug: 4wzuaiMEh5R1tzwT7CBNVncjMl1zLu3fmz4CvMLu+UQ=
Date: Tue, 30 Oct 2012 14:16:12 GMT
Transfer-Encoding: chunked
Connection: keep-alive

2d2e

.....}[o#Y..{..lNO..-..[...u.J...R.&.L&.....j....0.'...a.afoX.^`.{...3.`.{.....?._..L&/.....w.]...d.s.....'"...7.6N..[R...k_..?..
COMPRESSED CONTENT.....

Full Page Refresh

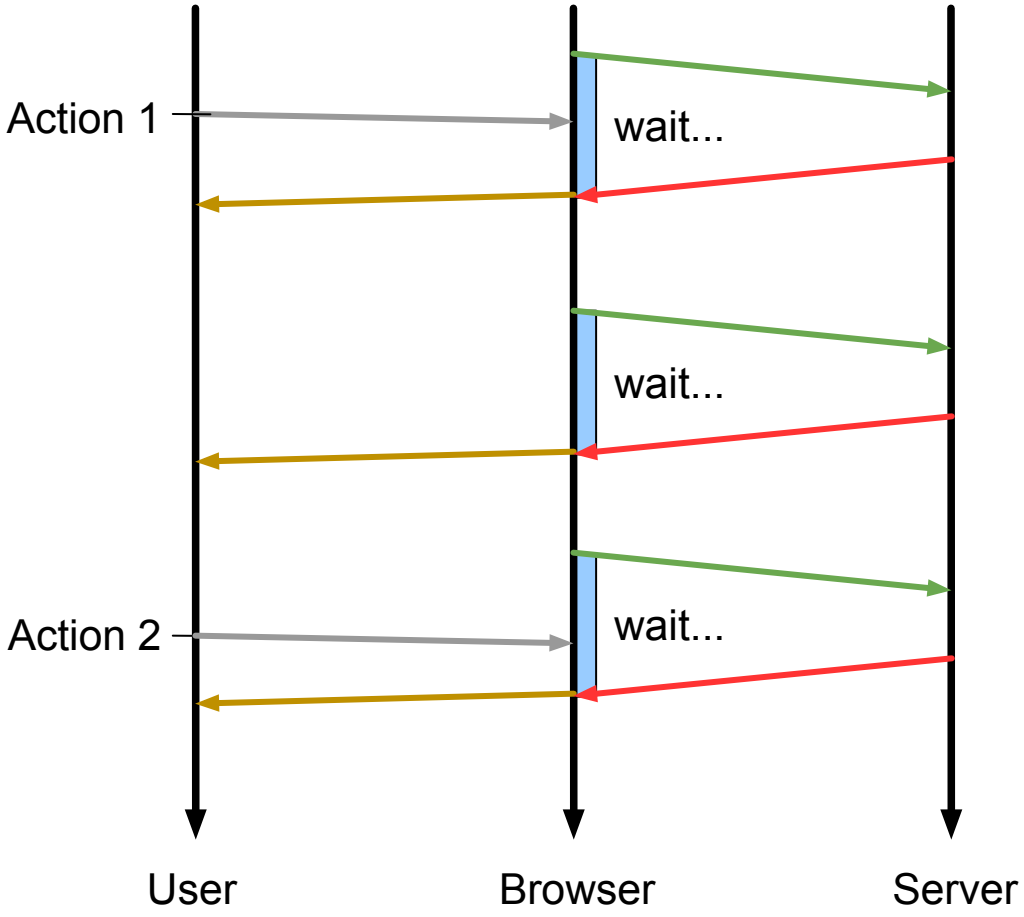


Typical issues:

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

***Sensor example**: for each refresh, the full HTML page with the current values is retrieved*

Ajax Polling



Typical issues:

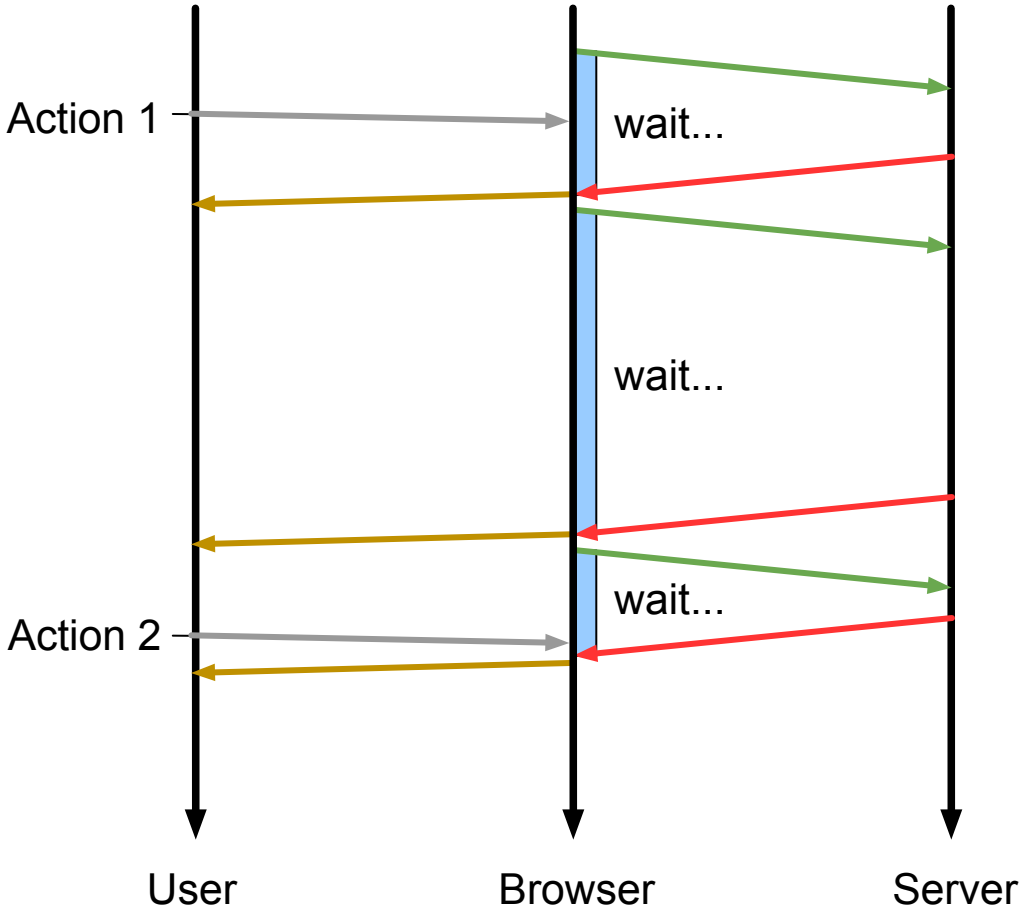
- Low update frequency; no real time
- High bandwidth usage (but lower than page refresh)
- High load on Web server

Advantages:

- User interface is never blocked

***Sensor example**: for each poll, the current values are retrieved*

Comet Long Polling (or HTTP Long Polling)



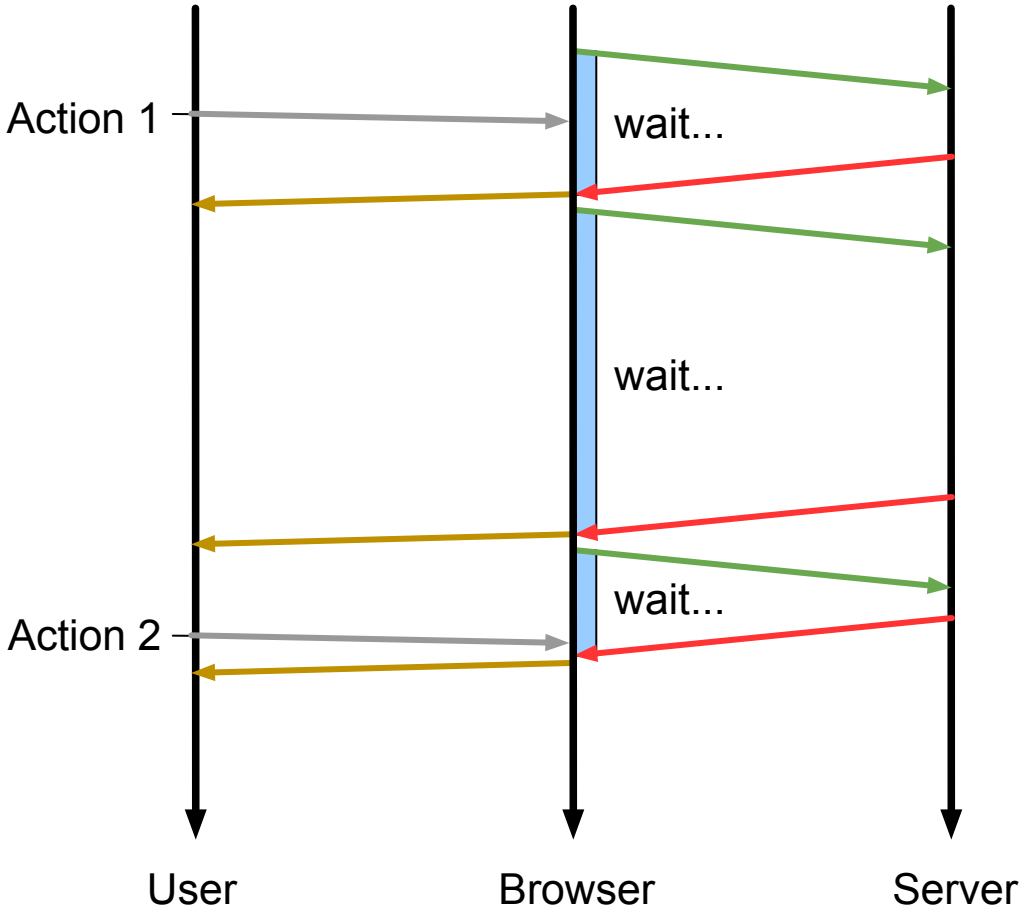
Typical issues:

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

Advantages:

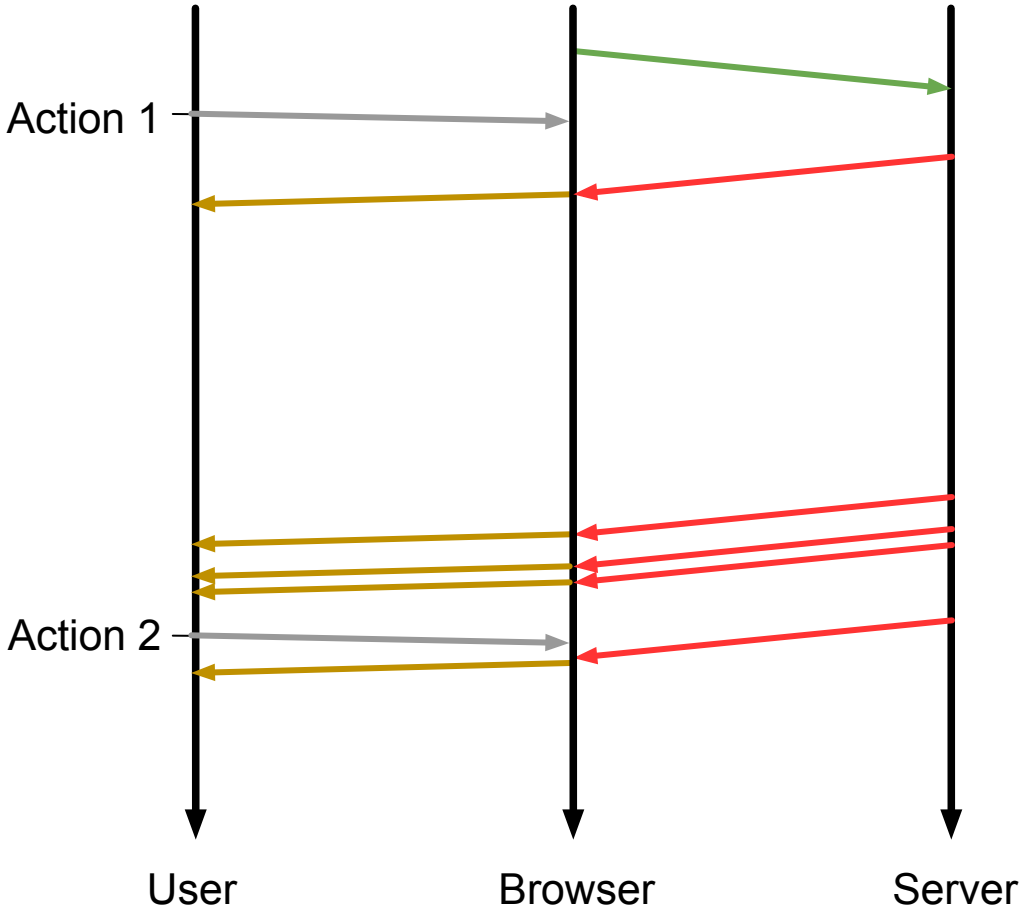
- User interface is never blocked
- Low latency on low-frequency events

Comet Long Polling (or HTTP Long Polling)



***Sensor example**: for each poll, the new values are retrieved only when they become available. Otherwise, the request is kept pending (long poll)*

Comet Streaming (or HTTP Streaming)



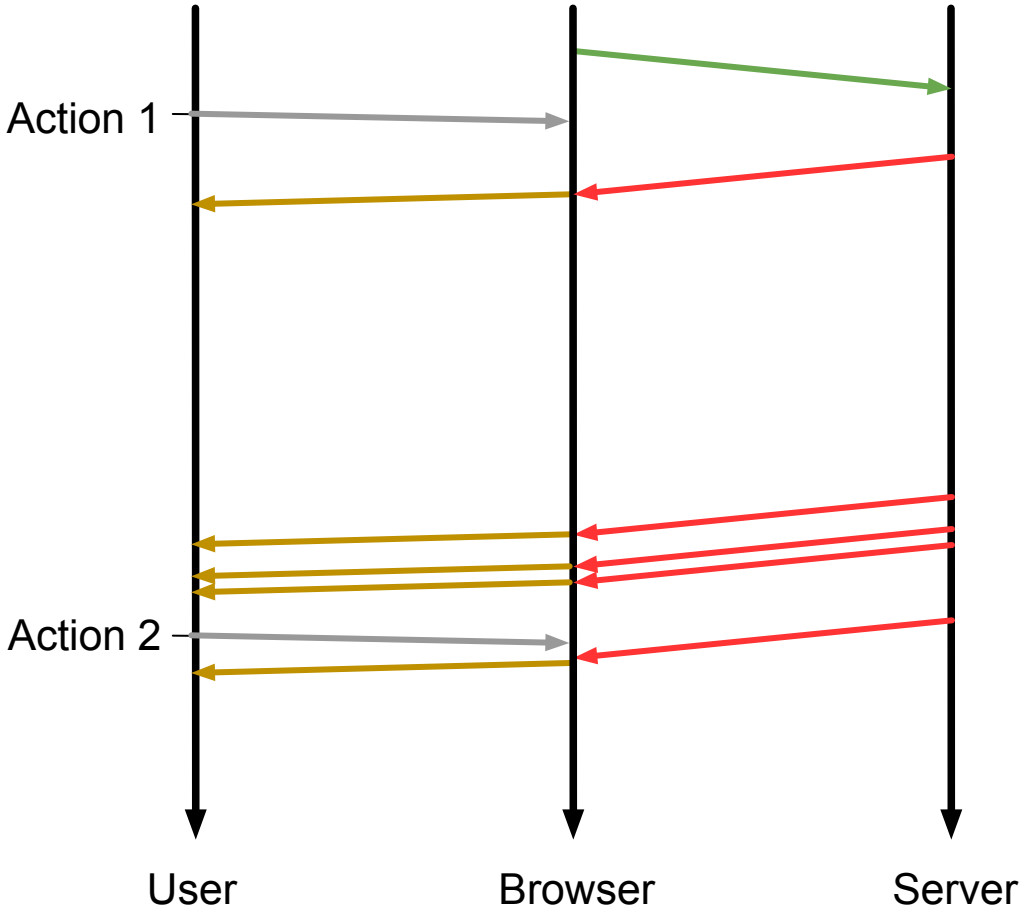
Typical issues:

- May be blocked by some anti-virus software mounted on proxy servers

Advantages:

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

Comet Streaming (or HTTP Streaming)



Possible techniques:

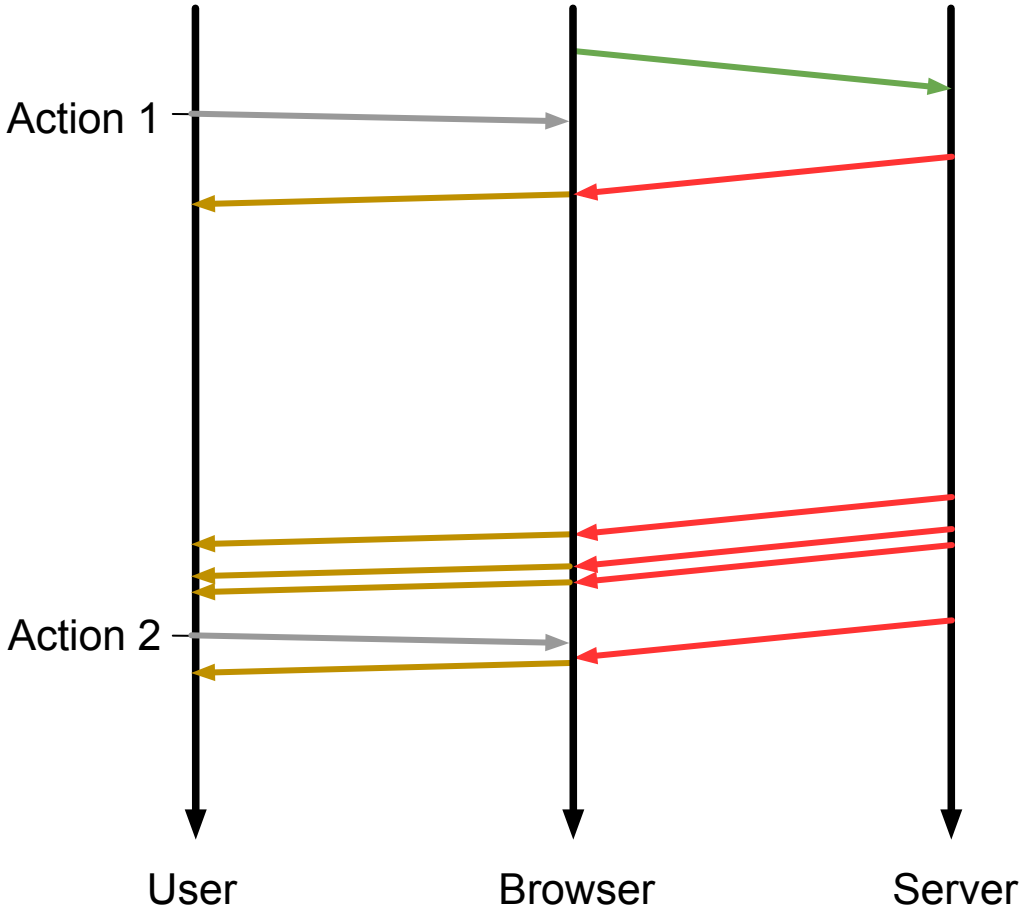
- Iframe streaming
- XHR streaming
- Flash streaming
- Server-Sent Events (SSE)

***Sensor example**: the server keeps pushing real-time updates as they become available, whatever is the frequency, without request/response round trips from the client*

Third Wave: WebSockets

- Goal:
  - Full-duplex asynchronous communication between a web client and a web server
- Why not just plain TCP?
  - Client runs untrusted code: origin-based security model; ports 80/443
  - WebSockets are message oriented (*onmessage*, *send*), TCP is stream oriented
  - WS split messages into frames, to allow:
    - Sending messages of unknown size without buffering
    - Multiplexing more logical channels on the same connection
    - Masking (XOR with random key) frames sent from the client to prevent cache poisoning on flawed proxy servers

WebSockets



Typical issues:

- There might be some firewall or proxy still blocking WebSockets

Advantages:

- Same as HTTP Streaming

***Sensor example**: it's a unidirectional scenario (from server to client), so with WebSockets the behavior is the same as with HTTP Streaming.*

WebSockets vs. HTTP/1.1

Myth:

WebSockets are better than HTTP for sending data from the server to the client (use less bandwidth, have lower latency, etc.)

Myth debunked:

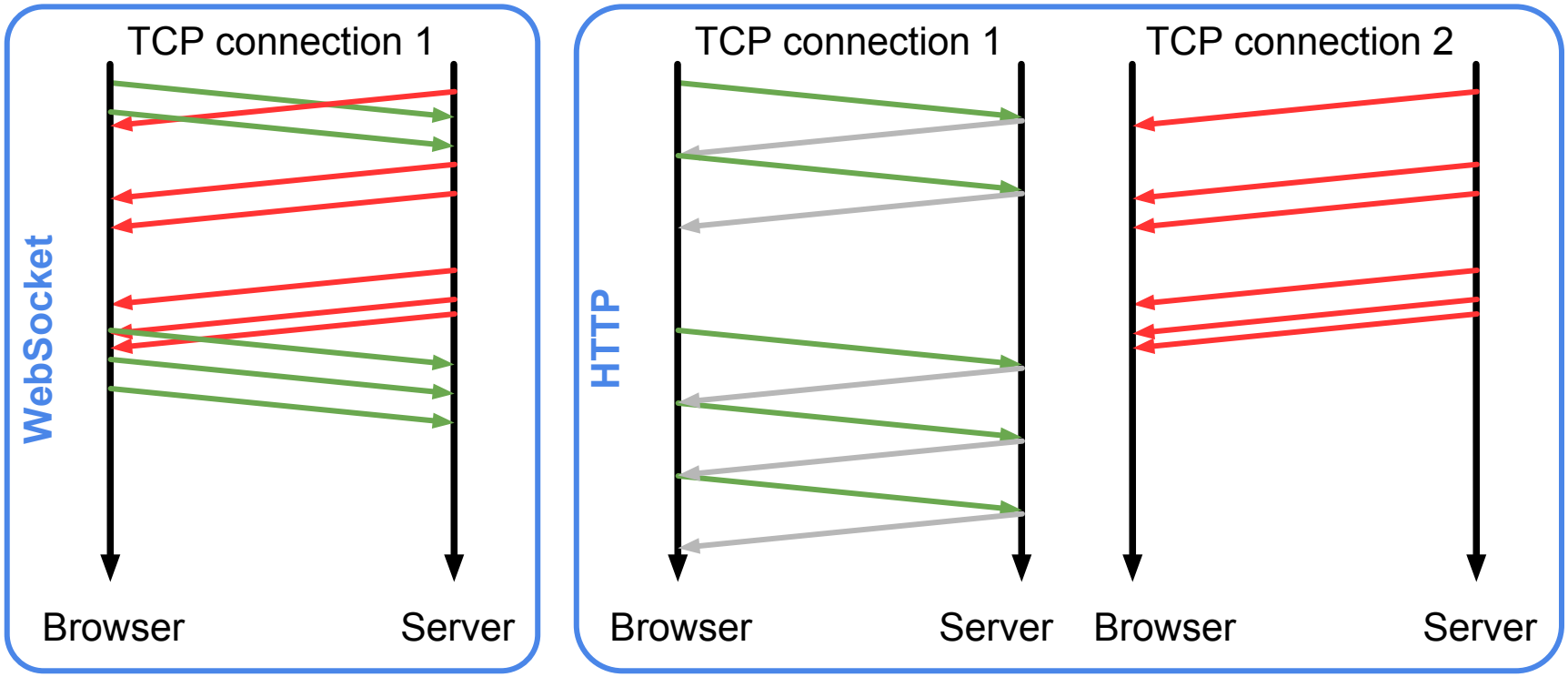
When sending data from the server to the client, WebSockets and HTTP Streaming behave exactly the same way.

After handshake, pure payload over TCP is streamed in both cases (WebSocket framing and HTTP chunking have a negligible difference)

WebSockets vs. HTTP/1.1

The real difference is for bidirectional scenarios:

1. HTTP requires at least 2 sockets
2. HTTP requires full round trip for each request (by default there is no pipelining)
3. HTTP gives no control over connection reuse (risk of a full SSL handshake for each request)
4. HTTP gives no control over message ordering



What About HTTP/2 ?

HTTP/2 improves performance over HTTP/1.1:

- Pipelining of requests
- Multiplexing of requests
- Compression of headers
- HTTP/2 server push

What is HTTP/2 server push?

- It is not a notification mechanism
- It can send page resources without waiting for requests
- It does not replace WebSockets
- It is possible to combine Server-Sent Events (SSE) and Multiplexing to "emulate" WebSockets (with no particular benefits)

In-app Notifications vs. Push Notifications

In-app Notifications (Data Streaming):

- Server sends real-time data directly to the app
- The app needs to be running to receive the data
- High throughput
- Low latency
- Guaranteed delivery

Push Notifications:

- Server sends notifications to Google/Apple servers
- Google/Apple servers send notifications to the device
- The app does not need to be running to get the data
- Low throughput
- No control over latency and actual delivery

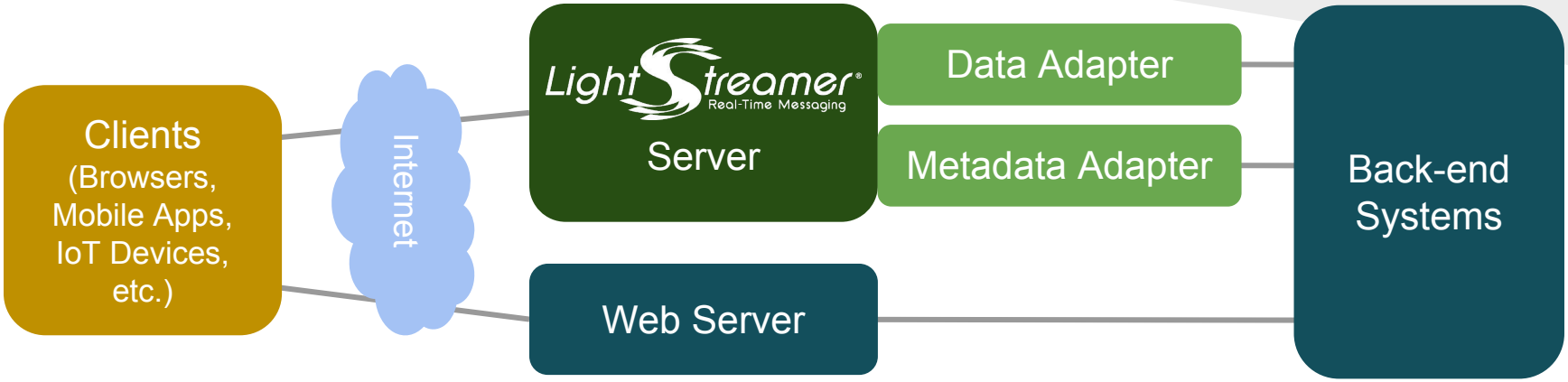
Lightstreamer Technology

What Is Lightstreamer?

Lightstreamer is a real-time message broker optimized for the Internet

- Implements WebSockets and HTTP Streaming/Polling
- Implements native push notifications
- Massively scalable
- Passes through any kind of network intermediary (firewalls, proxies, etc.)
- Supports any client-side platform
- Integrates with any back-end infrastructure
- Automatically throttles bandwidth
- Supports custom authentication and authorization
- World class track record

Lightstreamer Architecture



Lightstreamer Server: stand-alone process that runs in a Java virtual machine

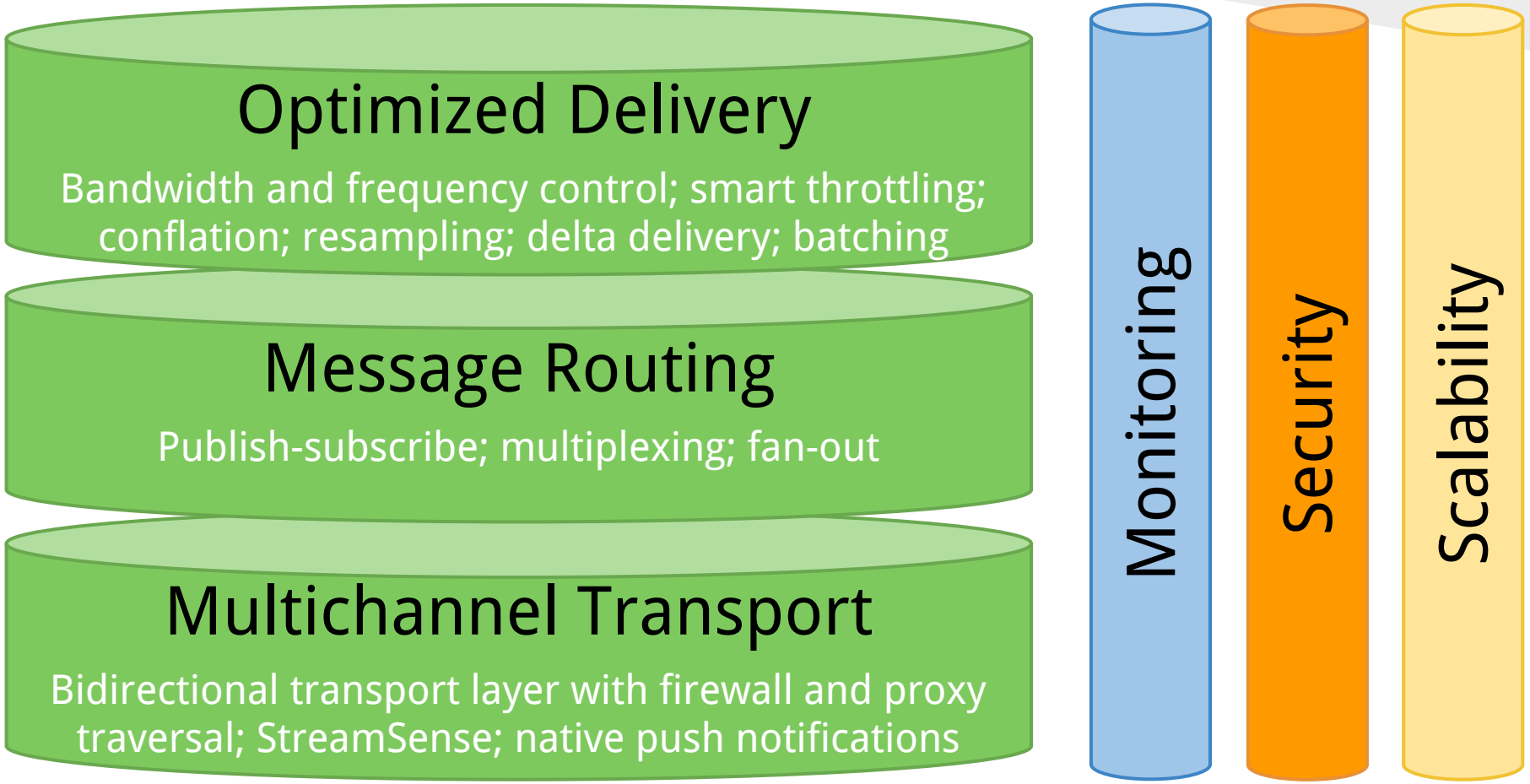
Lightstreamer Data Adapter: custom component based on the provided API (**Java**, **.NET**, **Node.js**, **Python**, and **TCP sockets**) that attaches to the data feed and injects the real-time data flow into the Server

Lightstreamer Metadata Adapter: custom component based on the provided API (as above) that manages authentication and authorization

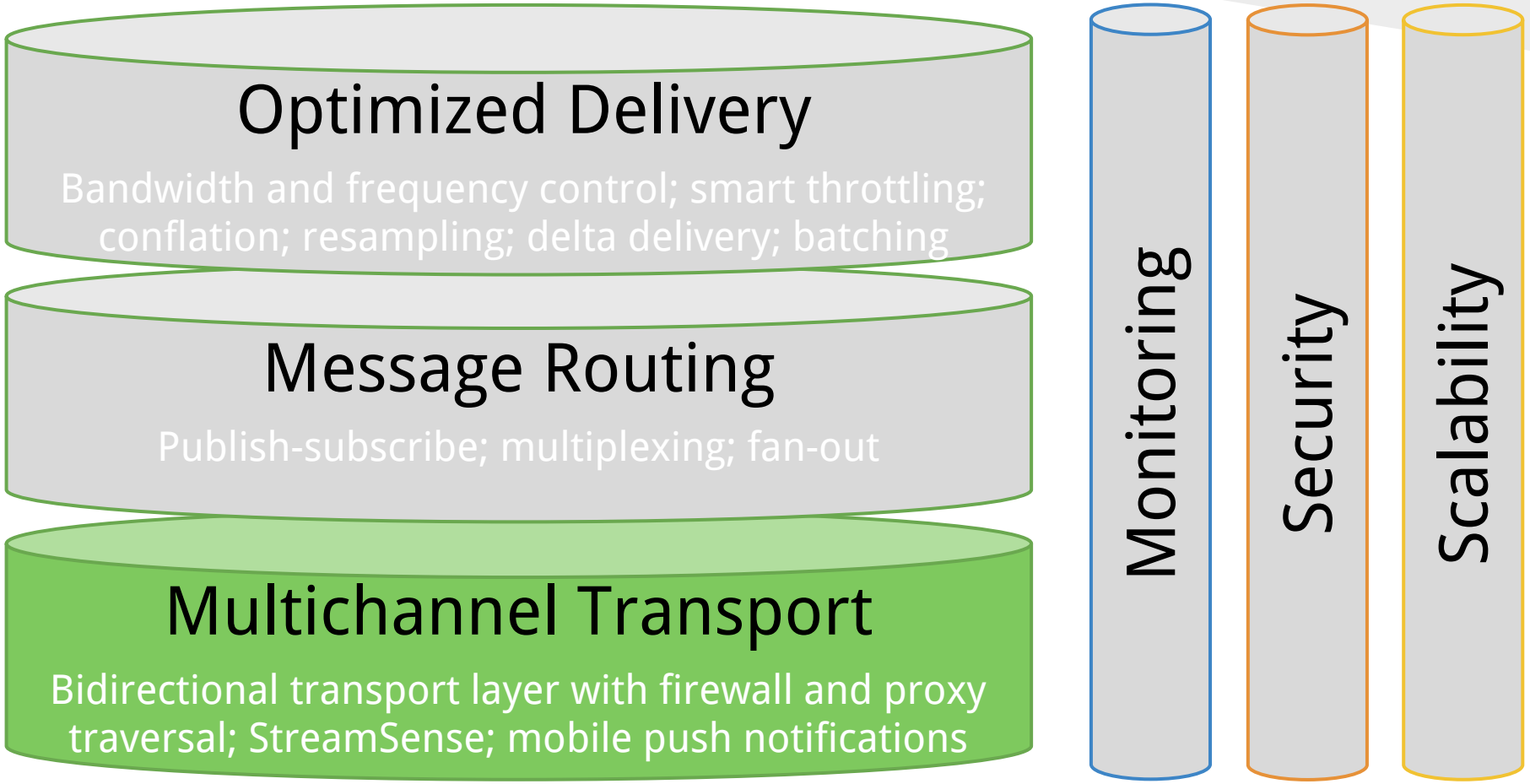
Rich Set of Lightstreamer Client APIs

- **Web** (compatible with any browser, including older browsers and mobile browsers; supports frameworks like **Angular**, **React**, **Vue**, as well as hybrid frameworks, such as **PhoneGap** and **Electron**)
- **Android**
- **Apple** (**iOS**, **macOS**, **tvOS**, and **watchOS**)
- **Microsoft** (**.NET** and **Excel**)
- **Java SE**
- **Node.js** (for both server-side code and **React Native** apps)
- **Python**
- **Unity**
- **Legacy** (**Adobe Flash**, **Flex**, **AIR**; **Silverlight**; **Java ME**; **BlackBerry**; **Windows Phone**)
- **Generic clients** based on the **TLCP open protocol**
  (see https://lightstreamer.com/docs/client_TLCP_base/TLCP%20Specifications.pdf)

Logical Layers of Lightstreamer Server

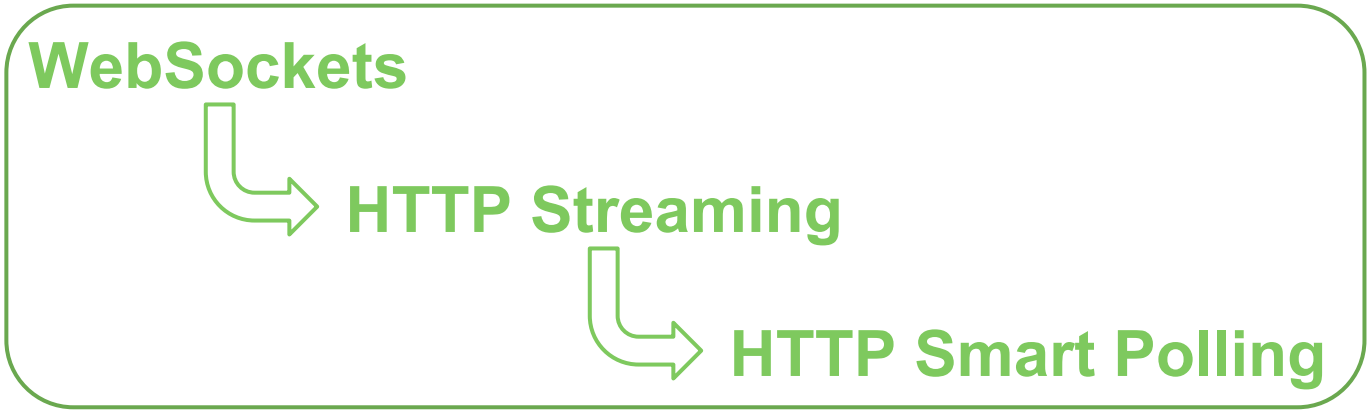


Logical Layers of Lightstreamer Server



Multichannel Transport: StreamSense

- Automatic and fast detection of the best transport on a per-client basis
- Upper layers are fully abstracted from the actual transport



Multichannel Transport: Bidirectional Channel

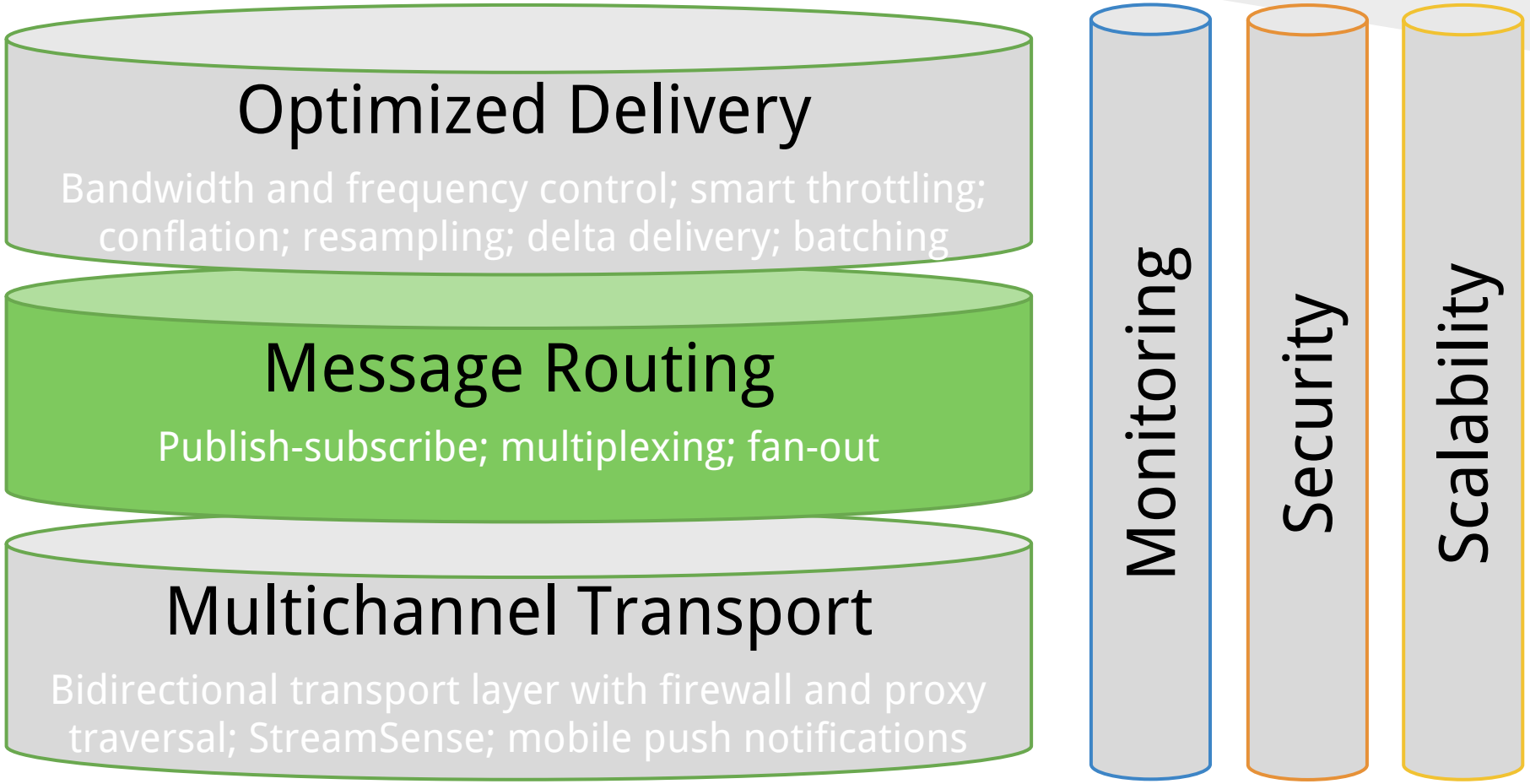
- Efficient and reliable bidirectional channel provided in all the cases (whatever is the underlying transport)
- Messages sent from the client to the server have in-order guaranteed delivery with automatic batching
- Lightstreamer enriches HTTP:
  - Messages are acknowledged explicitly
  - Lost messages are retransmitted automatically
  - Out-of-order messages are reordered automatically
  - Underlying socket is kept open for reuse via reverse heartbeats
  - Multiple requests are automatically batched, to highly reduce the number of HTTP round trips

See live Round-Trip Demo:
https://demos.lightstreamer.com/RoundTripDemo/

Multichannel Transport: Push Notifications

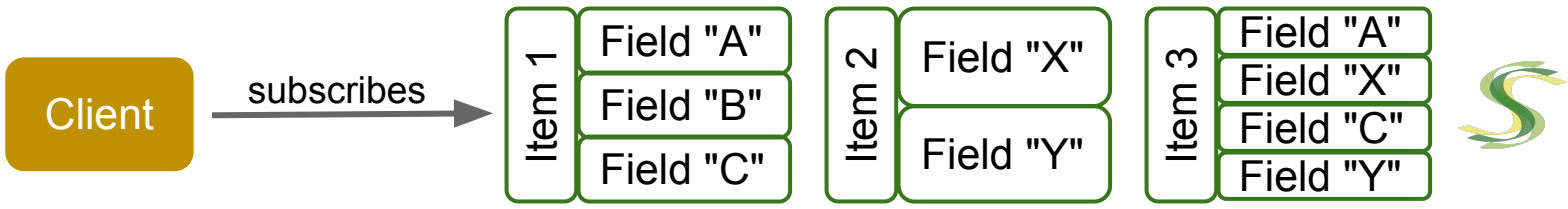
- Data can be delivered to mobile clients using native push notifications too
- Apple APNs and Google FCM are supported
- If an app is not active, the device will receive live updates in any case
- No extra development on the server side: messages originated from the Data Adapter can use both the streaming channel and the push notification channel
- Trigger support: determine what messages should result in a push notification

Logical Layers of Lightstreamer Server

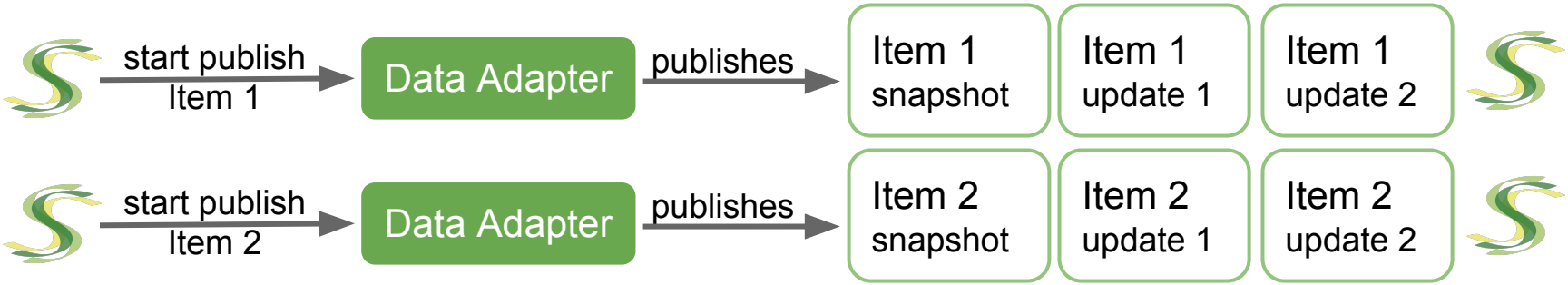


Message Routing: Publish-Subscribe

- Client **subscribes** to items with schemas (sets of fields):

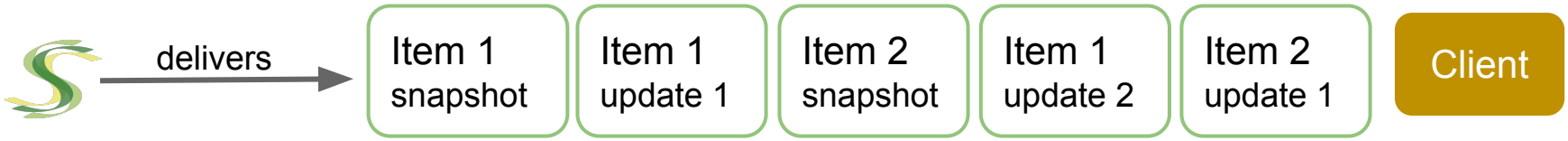


- Data Adapter **publishes on demand**:

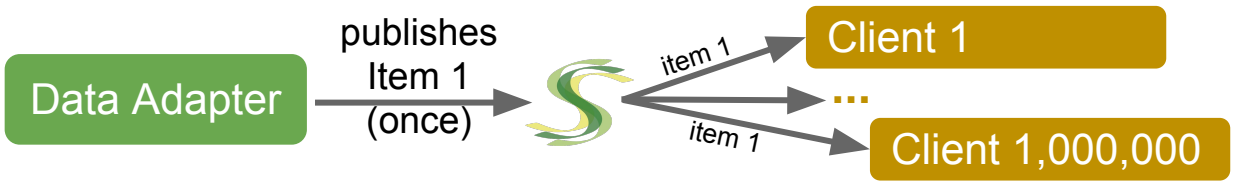


Message Routing: Publish-Subscribe

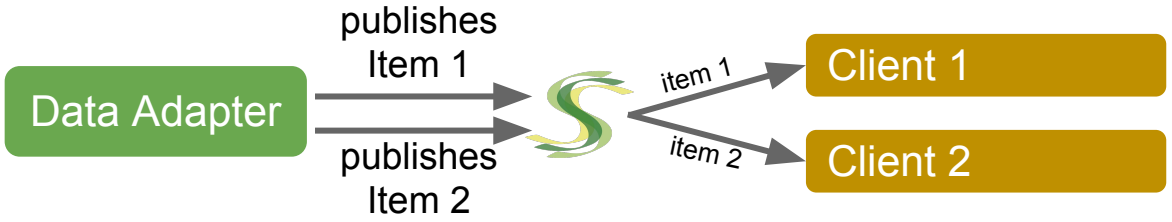
- Server sends multiplexed data to Client:



- Any routing scenario is supported (broadcast, multicast, unicast):



Massive fan-out, broadcast



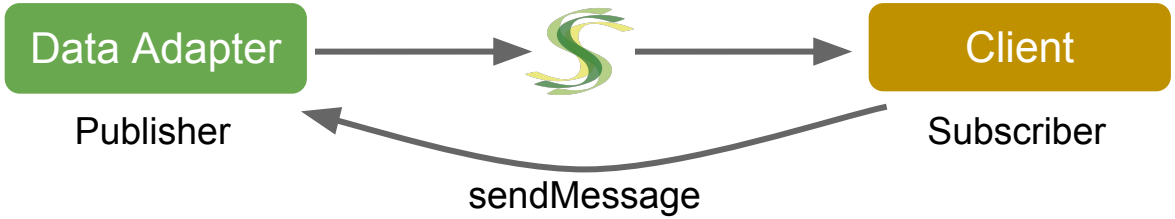
Personal messages, unicast

Message Routing: Publish-Subscribe

- Asymmetric pub-sub:

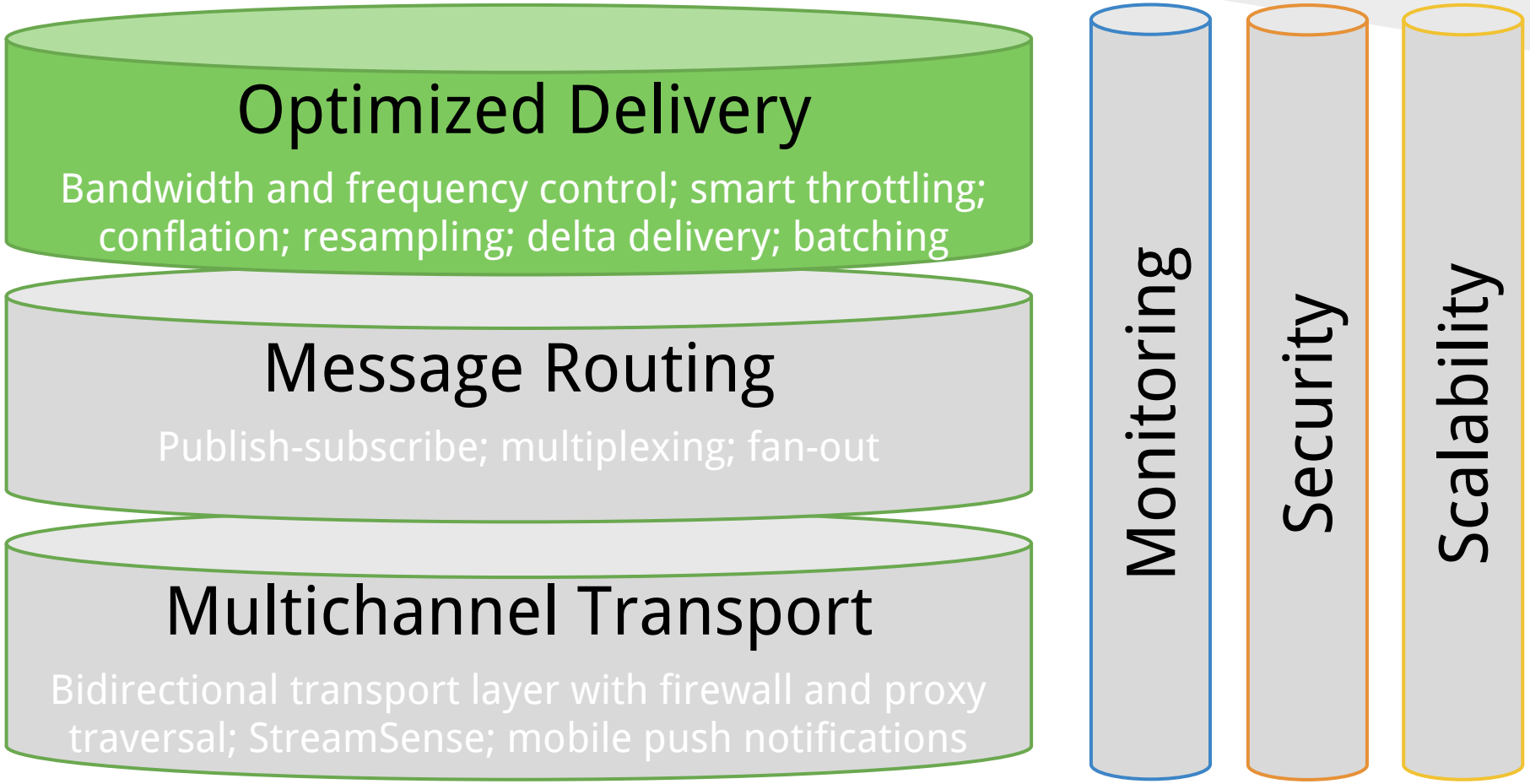
  - In many scenarios the "data feed" is completely different from the data consumer (topology, protocol, business model)
  - Optimization for massive publishing from server-side data feeds

- Clients can still publish:



  - The Client (Subscriber API) can send messages to the Adapter to be processed and possibly incorporated into the data stream

Logical Layers of Lightstreamer Server



Optimized Delivery: Filterability

- Data filterability
  - Based on the nature of the data, series of updates to an item can be filtered, to reduce frequency, via:
    - Queueing
    - Resampling
    - Conflation
- Lightstreamer's filtering
  - For each subscription of each client, Lightstreamer allows to define how data can be filtered, with several parameters
  - Filtering is then applied on the fly to the data stream based on a number of static and dynamic conditions

Optimized Delivery: Throttling

- Bandwidth Control
  - For each client, a maximum bandwidth can be allocated to the multiplexed stream connection
- Frequency Control
  - For each subscription of each client, a maximum update frequency can be allocated
- Smart Throttling
  - Internet congestion is detected

Lightstreamer heuristically combines these three variables to dynamically throttle the data flow with filtering

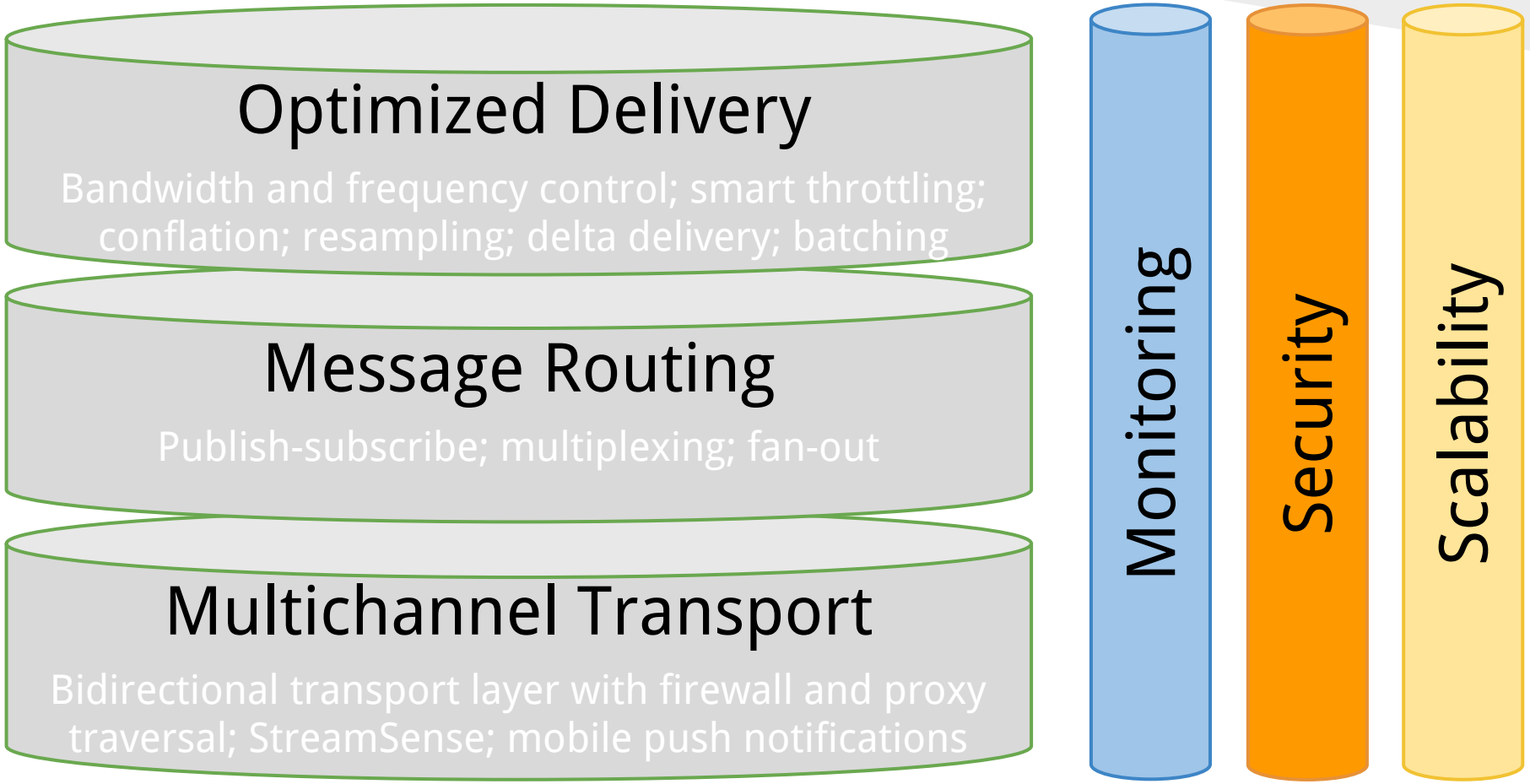
See live Bandwidth and Frequency Demo:
https://demos.lightstreamer.com/BandwidthDemo/

Optimized Delivery: Other Mechanisms

- Batching and TCP packet optimization:
  - Data is aggregated efficiently within TCP packets
  - Configurable trade-off between latency and overhead reduction, overriding Nagle's algorithm
- Lightweight protocol:
  - Position-based protocol with negligible overhead (no JSON, no XML, no metadata redundancy)
- Delta delivery:
  - For subsequent updates to an item, only the actually changed fields (delta) are sent; custom selectors available
- Multiple subscription modes:
  - MERGE, COMMAND, DISTINCT, RAW

See live Market Depth Demo:
https://demos.lightstreamer.com/MarketDepthDemo/

Logical Layers of Lightstreamer Server



Scalability

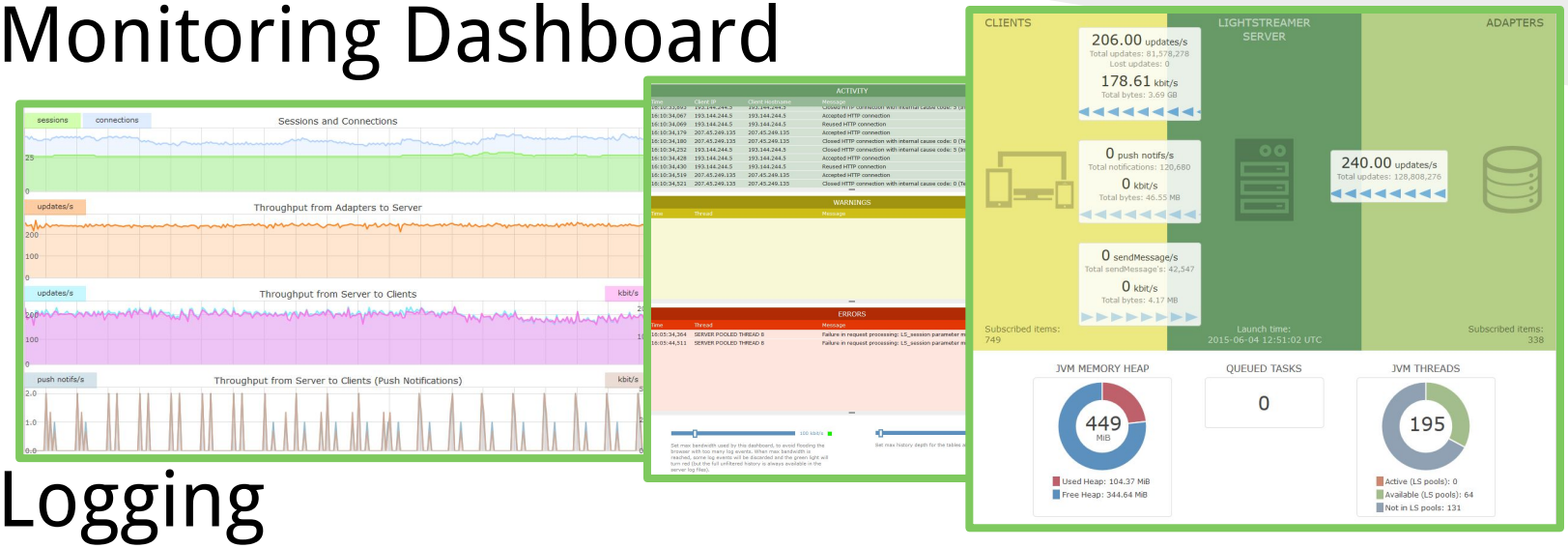
- Concurrent staged event-driven architecture
  - Non-blocking I/O used for all types of connections
  - Graceful degradation of the quality of service
  - Tested on a single box with:
    - One million connections with low frequency traffic
    - Tens of thousands of connections with very high frequency traffic
- Vertical scalability
  - An instance of Lightstreamer Server can fully leverage multiple CPUs and cores available in a box
- Horizontal scalability
  - Clustering via any standard Web Load Balancer

Security

- Secure connections
  - WSS and HTTPS support based on SSL/TLS strong encryption and configurable cipher suites
  - Server-side and client-side certificate support
- Authentication
  - Credentials are received from the client, together with HTTP headers and connection properties
  - Custom Metadata Adapter validates them
- Fine-grained authorization
  - Every subscription and QoS request done by the clients is authorized through the Metadata Adapter

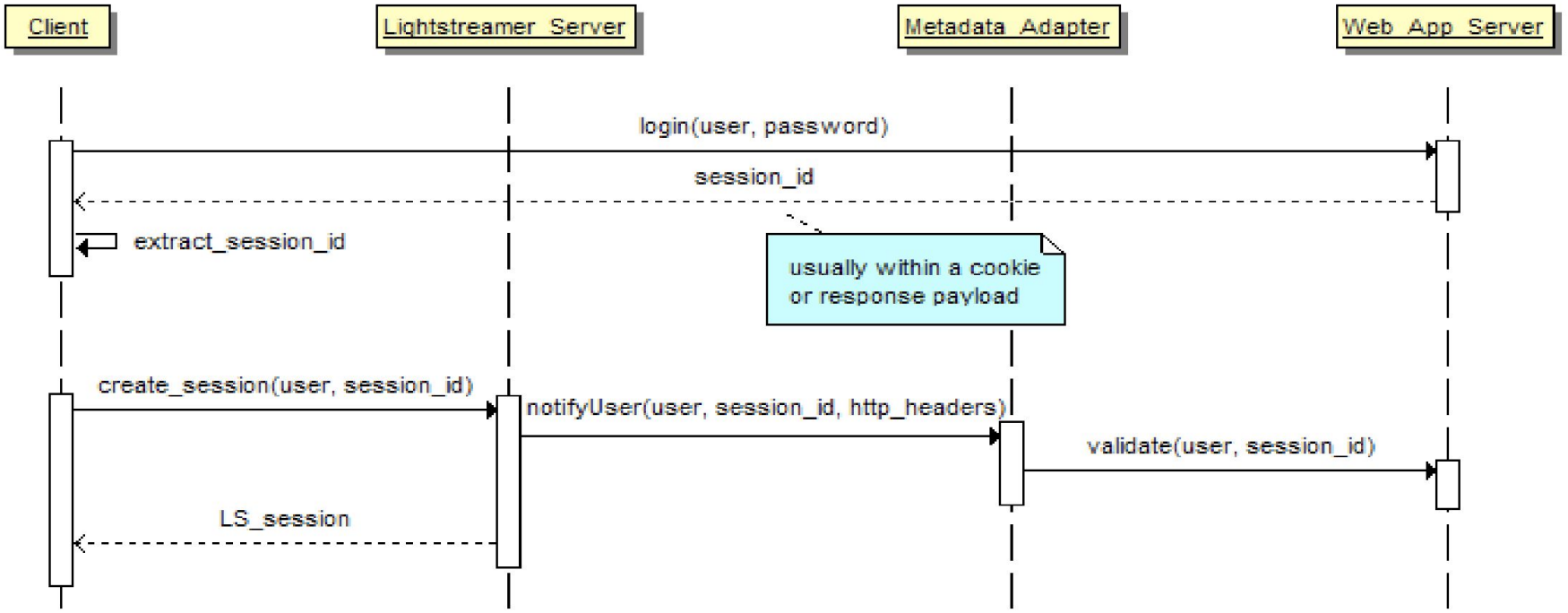
Monitoring

- Monitoring Dashboard

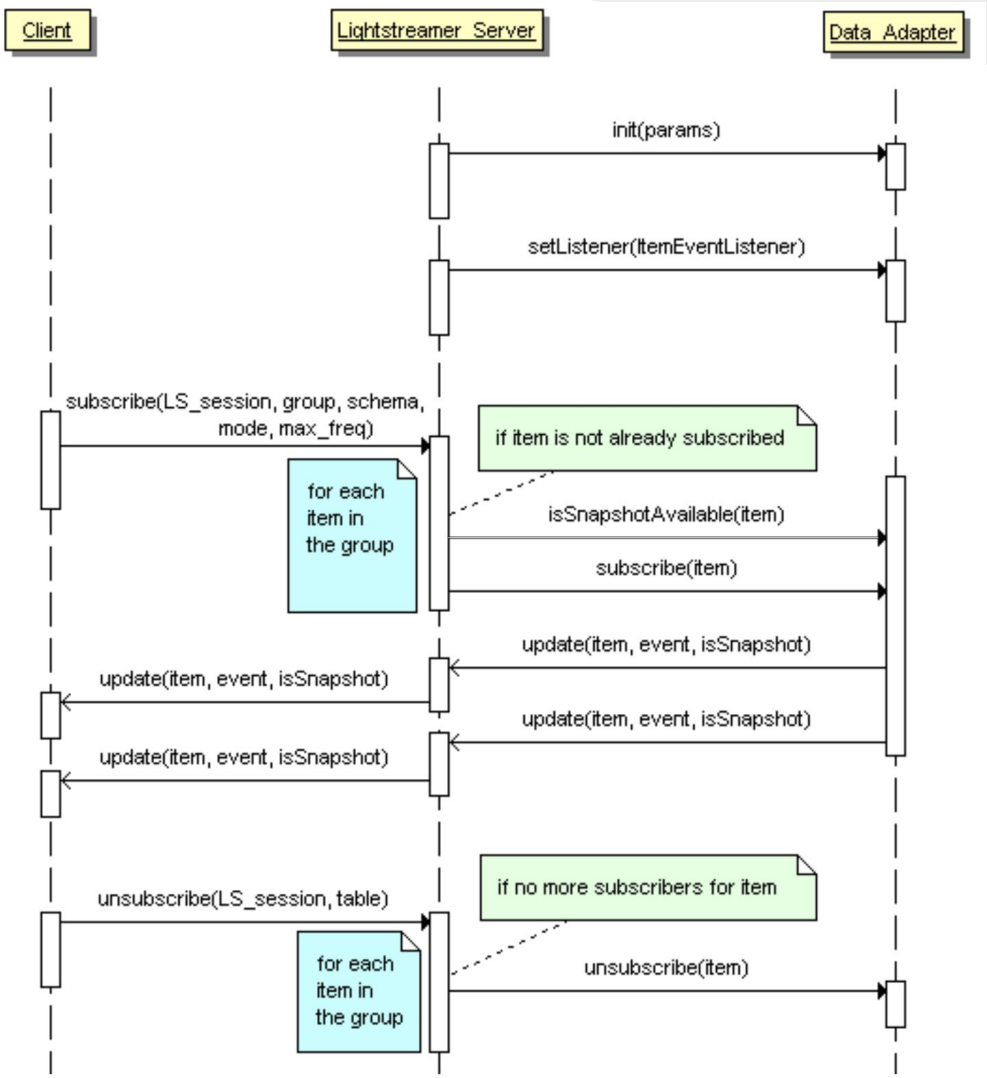


- Logging
  - Fine-grained configurable logging, with several categories, levels, and appenders
- JMX
  - Extensive metrics exposed via a JMX interface, to hook into application management facilities

Metadata Adapter Model: Authentication



Data Adapter Model: Data Push



Lightstreamer Success Stories

Some Lightstreamer Customers

Morgan Stanley

Fidelity INVESTMENTS

Fannie Mae

IG

DNB

DELTA

UBS

OOCL

INTESA SANPAOLO

DEUTSCHE BÖRSE GROUP

FOREX.com

eToro

bwin

GoldBet

sky

Way More Customers...

300+ companies using Lightstreamer from 6 continents.

Thousands of servers installed.

Millions of end users served.

Lightstreamer Named a "Cool Vendor" by Gartner

Gartner, "Cool Vendors in Application and Integration Platforms, 2012", by Massimo Pezzini and Jess Thompson, 11 April 2012.

Cool Vendor Report 2012 cites Weswit (former name of Lightstreamer company), with its Lightstreamer product, as innovative, impactful and intriguing in the area of Application and Integration Platforms.

"Web streaming is an emerging form of MOM aimed at enabling back-end applications to send real-time messages over the public Internet, typically to large numbers (up to millions) of mobile or stationary endpoints, according to a publish-and-subscribe model". When analyzing 'Who should care' the report goes on to explain: "ISVs, SIs and cloud service providers that require efficient, low-latency and scalable publish-and-subscribe data distribution to mobile and Web-based endpoints should look at Web-streaming technologies as a way to add value to their offerings by enabling reliable and relatively easy-to implement connectivity."

Outstanding Customer Satisfaction

> *As Lightstreamer is pivotal in our overall offering, the Lightstreamer support team was a key factor in our success. For its reputation in the financial industry, choosing Lightstreamer was a no-brainer for us.*

Israel Kalush, VP Engineering, eToro

> *Lightstreamer has been a very stable and hassle-free piece of infrastructure that also has made our client programming life easier. We have operated a mission critical streaming system for nearly 3 years with a minimum of operational trouble.*

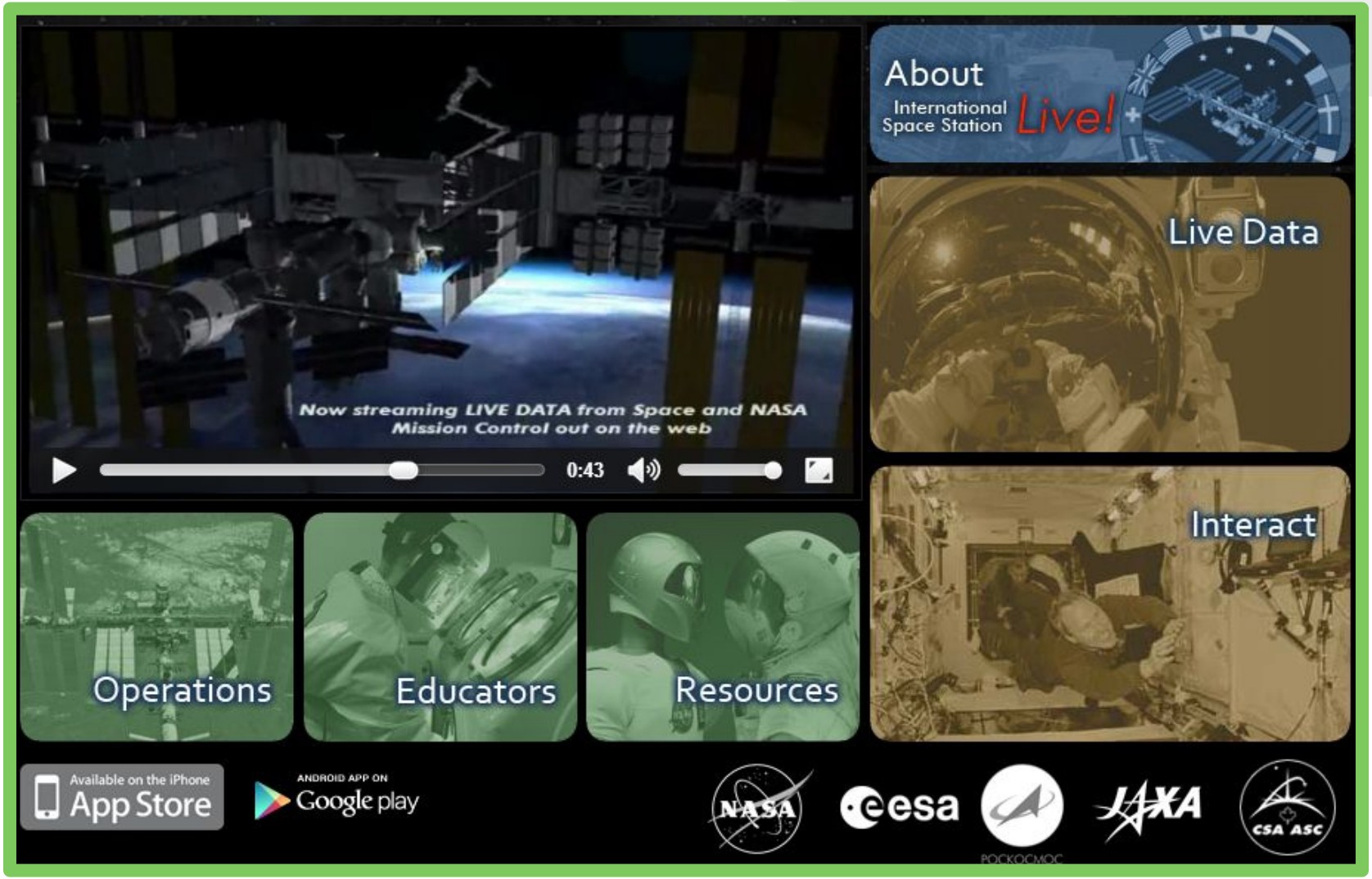
Oddmar Sandvik, DNB

> *Our experience of Lightstreamer support has been very positive, they have set a high bar for other vendors to strive to achieve. They are very proactive in helping to resolve issues.*

Ivan Gowan, Head of IT Development, IG

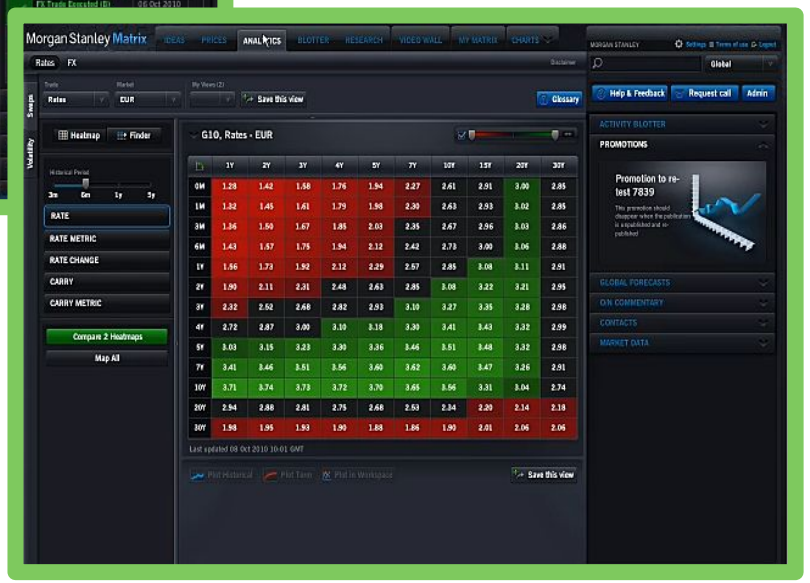
NASA: International Space Station Live

www.isslive.com



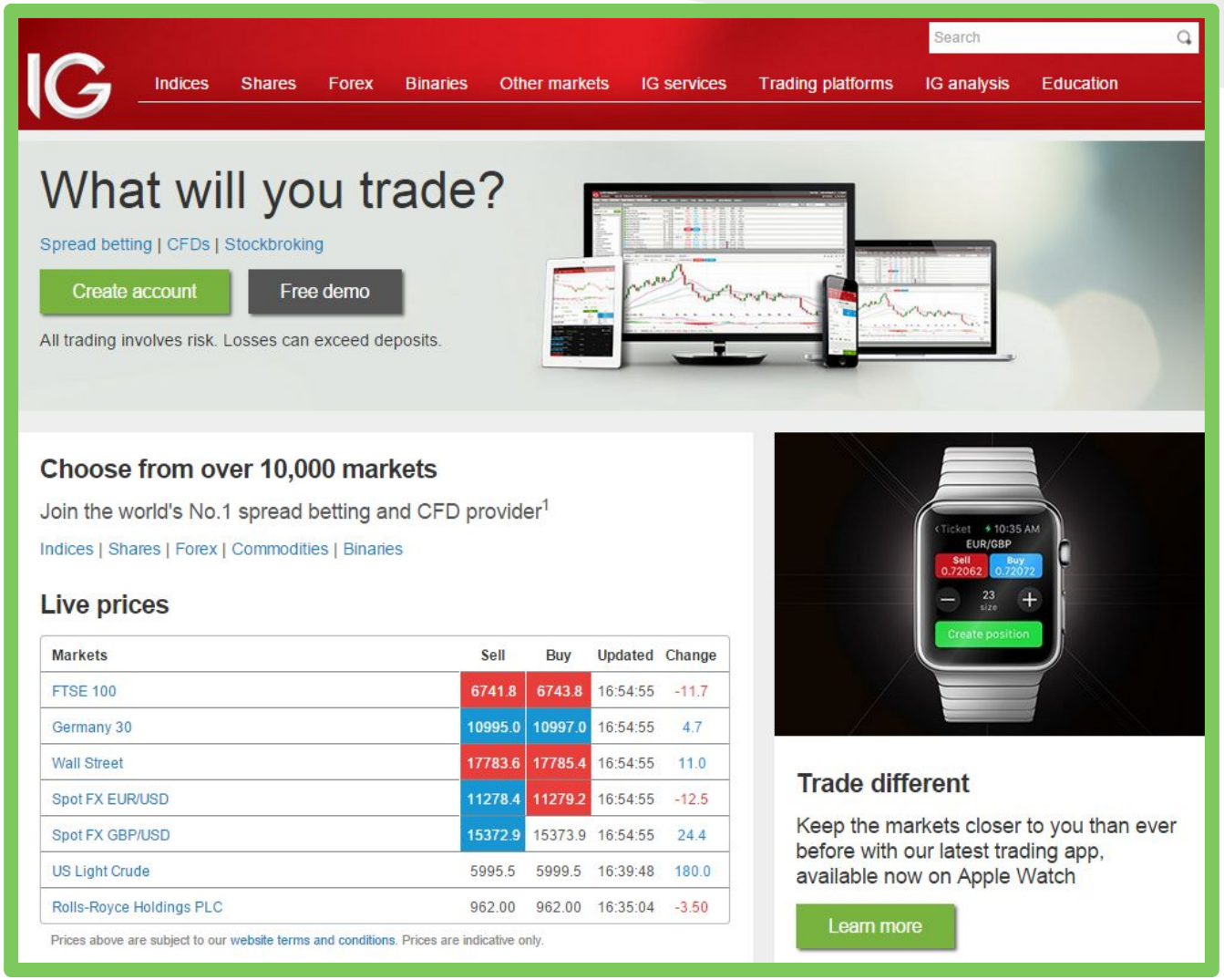
Morgan Stanley: Matrix

www.morganstanley.com/matrixinfo



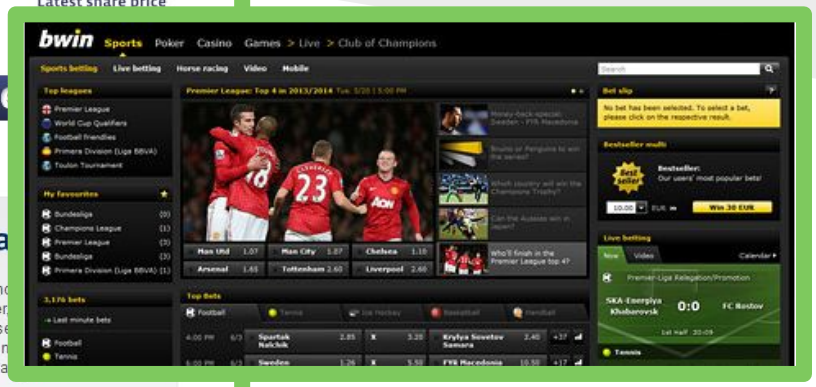
IG: Spread Betting and CFDs

www.ig.com



bwin.party: Sports Betting and Online Gaming

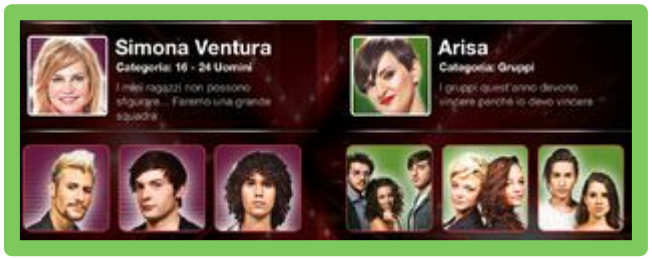
www.gvc-plc.com



GVC Holdings

"X Factor" TV Show: Remote Clapping and Voting

xfactor.sky.it



MQTT.Cool and JMS Extender

Other Products Based on the Lightstreamer Engine

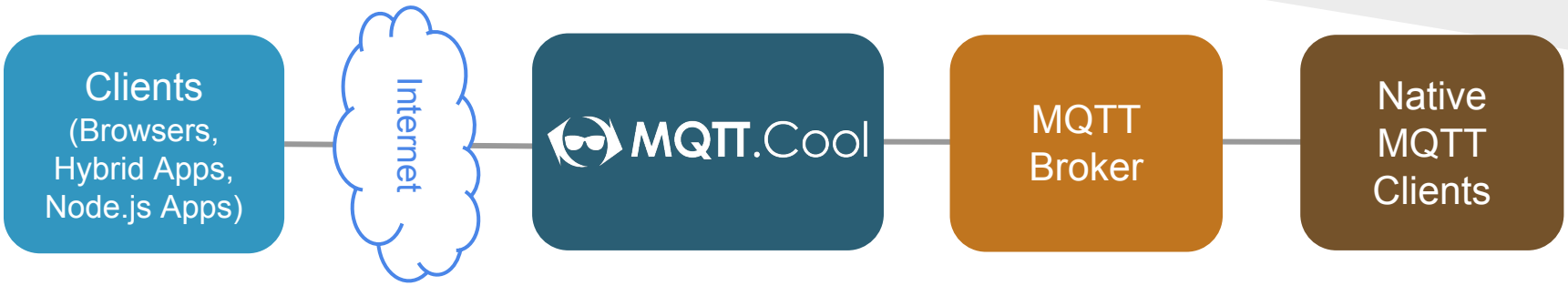
Lightstreamer is used as the core for two specialized products:

- MQTT.Cool
- JMS Extender

These are gateways that extend two messaging protocols (MQTT and JMS) over the web

Any legacy JMS solution or any new IoT/MQTT platform can easily get browser-based clients

MQTT.Cool



MQTT.Cool extends any third-party MQTT broker with new out-of-the-box features. Any web page running inside a web browser will instantly become an MQTT client, ready to send and receive real-time MQTT messages through firewalls and proxies.

- On the client side, a Paho-like API is provided as part of the JavaScript client library
- On the server side, a Java hook API is provided to implement custom authentication and authorization

MQTT.Cool Improves Any MQTT Broker

Security

- Authenticate users with total flexibility
- Add fine-grained authorization
- Offload TLS/SSL encryption
- Increase security by avoiding direct access to the broker

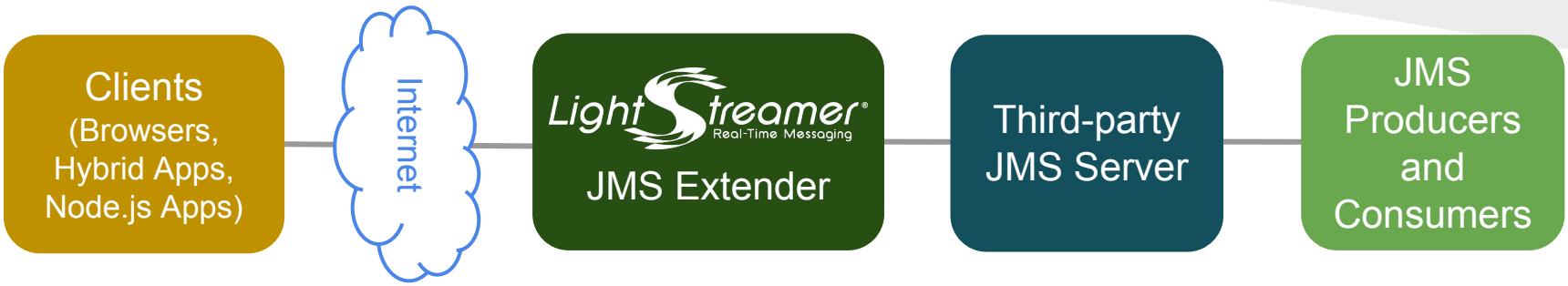
Architecture

- Connect to any MQTT broker from anywhere on the Internet
- Develop web clients with friendly Eclipse Paho-like API
- Access multiple MQTT brokers with a single connection

Performance

- Scale up any MQTT broker with massive fan-out
- Always receive fresh data with adaptive throttling and conflation
- Get full control over bandwidth and event frequency

JMS Extender



Lightstreamer JMS Extender leverages the Lightstreamer technology to *extend* any third-party JMS server by:

- Extending the JMS connections from the LAN into the web
- Extending the JMS API from Java to JavaScript
- Extending the JMS server scalability
- Extending the JMS security model

Web pages can exchange messages with legacy JMS applications through the Internet with no security issues

Benefits of JMS Extender

- Connect to any JMS server from the Internet
  Even from behind the strictest corporate firewalls
- Use the JMS API in your JavaScript code
  Full JMS API in any web browser, as well as any Node.js application
- Massively scale out your existing JMS Server
  Offload connection fan-out to the JMS Extender
- Add fine-grained authentication and authorization
  The Hook API enables to implement custom security rules
- Connect to multiple JMS servers
  A single JMS Extender can connect to different JMS servers
- Support for advanced JMS features
  Multiple acknowledge modes, once-and-only-once message delivery, etc.

Follow Lightstreamer

Website: lightstreamer.com

Blog: blog.lightstreamer.com

Forums: forums.lightstreamer.com

Newsletter: eepurl.com/dv10w1

 @Lightstreamer

 github.com/Lightstreamer

 facebook.com/Lightstreamer

 linkedin.com/groups/2218807