weswit
Software Made in Italy


# Clustering for Load Balancing and Fail Over

Last updated: 01/03/2013

# Table of contents

# 1 Introduction

Clustering of Lightstreamer Servers is achieved through a standard Load Balancer (LB), such as products by Cisco, F5, Crescendo Networks, etc., which in a production scenario is probably already in use for the Web Server farm. A hardware LB (appliance) is highly preferred over a software one, because the Lightstreamer Server should "see" the actual TCP flow that reaches the Client, without intermediate buffers, in order to throttle the bandwidth and manage network congestions.

The Lightstreamer Clients can use two different level-7 protocols to communicate with Lightstreamer Server: **HTTP** and **WebSocket**. In case the WebSocket protocol is not supported by the LB, Lightstreamer automatically falls back to HTTP. The LB should always be configured to make HTTP (or HTTPS) work seamlessly. Being able to support WebSocket too is recommended, but not required.

The Lightstreamer Client creates two categories of connections:
- **S**: *The session creation connection*, which is the first connection in the lifecycle of a Lightstreamer client session. It is always a short-lived HTTP(S) POST request, by which the client authenticates and gets its client-id. Upon unexpected session termination (usually due to a disconnection), another "S" type connection is done by the client. These connections can be always freely balanced (with no stickiness requirements).
- **CR**: *The control and rebind connections*, which are used for sending subcriptions and other commands and upon which the actual data flow is delivered. These connections require to be sticky to the node that has served the corresponding session creation connection. These connections can be based either on HTTP(S) or WebSocket(S).

Therefore, the cluster configuration has to ensure the stickiness property.

Usually, the LB should be configured to switch traffic at level 7 (HTTP). But there are cases (especially with WebSocket) where configuring traffic switch at level 4 (TCP) enhances compatibility. In some cases, using level 4 loses information on the originating client IP, which might be otherwise added as an HTTP header when using level 7. But if level 4 acts as a real low-level switch, than the originating IP will be kept.

Browser-based clients may pose additional constraints, due to the origin checks performed by the browsers to ensure page interoperability.
First of all, in order to avoid any security warning originated by the browser, all the connection types should use the same protocol (i.e. **HTTP** or **HTTPS**).
Further constraints may involve the hostname to be used to access the Server through the Load Balancer. This is related to the presence of the **document.domain** setting on the client page: if set, the Server hostname must be consistent with the domain setting and this also holds for the public hostname of any Server instance (see the cases below).
The JavaScript Client Guide document clarifies when setting the **document.domain** property may be advisable.

The cases below refer to both browser-based clients and application clients (i.e. based on Java, .NET, etc.). The configuration of the client is like the following (based on Lightstreamer JavaScript SDK version 6.0 or newer):

```
var lsClient = new LightstreamerClient(
                          "http://push.mycompany.com",
                          myAdapterSet);
    ...
lsClient.connect();
```

To illustrate the available options for the cluster configurations, let's suppose that the Lightstreamer Server has been deployed on two boxes (so that the cluster contains two nodes). All the different

solutions explained in the following chapters can be naturally extended to the cases where more nodes are used. Fictitious public and private IP addresses and hostnames will be used in the examples.

# 2 HTTP-based Scenarios

## 2.1 Leverage LB Stickiness – Options 1.A

The "**sticky**" property of the Load Balancer is leveraged. Usually Load Balancers offer at least two types of stickiness mechanisms, which leads to two distinct options:

### 2.1.1 Option 1.A.1 – IP-address based

The Balancer routes the requests based on the source IP address. After a request from a new IP address is received, all the subsequent requests from the same IP address are routed to the same node chosen for the first request.
**Note**: Please bear in mind that some Internet providers exist that dynamically change the source IP address throughout different requests coming from the same Client. So the IP-address based mechanism is not always reliable, making the "cookie-based" one preferable.

The network configuration is like the following one:

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S, CR | push.mycompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 1 or LS Server 2 (balancing + sticky) | 10.0.0.1 or 10.0.0.2 | 80 |

The lightstreamer_conf.xml file will be the same for both the Servers and will contain the following element:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
```

No clustering configuration through the `<control_link_address>` element is necessary.

**WebSocket**: If the Balancer does not support WebSocket, it is suggested to configure it at level-4 (TCP), rather than at level-7 (HTTP).

### 2.1.2 Option 1.A.2 – Cookie based

The Balancer must support a "Sticky Cookie-Insert" mode, that means that the a cookie is inserted by the Balancer in the responses. Subsequent requests from the Client will send the cookie back, so that the Balancer will be able to correctly route them.

The network configuration is like the following one:

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S, CR | push.mycompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 1 or LS Server 2 (balancing + sticky) | 10.0.0.1 or 10.0.0.2 | 80 |

The lightstreamer_conf.xml file will be the same for both the Servers and will contain the following element:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
```

No clustering configuration through the `<control_link_address>` element is necessary.

**WebSocket**: As this option works at level 7 (level 4 is impossible because it does not allow cookie insertion), it is required that the Balancer supports WebSocket.

#### 2.1.2.1 Cookie based stickiness and non-browser clients

In order for the technique of inserting the “Sticky Cookie” to be successful, it is required that the client application manages cookies according with RFC 2965 or 6265. In this way, session cookies set by the Load Balancer will be sent back by the client upon any further connection to the same domain. This is normal behaviour for browsers and all applications running in a browser context can take advantage of that.

For standalone applications, this property may have to be enforced at application level.
The table below resumes the requirements for various cases, based on the different Lightstreamer Client SDK and execution context.

| SDK | cookie handling | notes |
|---|---|---|
| JavaScript | OK | Managed by the browser |
| Java SE | TO BE ENFORCED | Just issue: java.net.CookieHandler.setDefault(new java.net.CookieManager(null, java.net.CookiePolicy.ACCEPT_ALL)); |
| Android | TO BE ENFORCED | Just issue: java.net.CookieHandler.setDefault(new java.net.CookieManager(null, java.net.CookiePolicy.ACCEPT_ALL)); |
| .NET | OK | Managed by LS library |
| Silverlight | OK | Managed by the browser |
| Windows Phone | OK | Managed by LS library |
| WinRT | OK | Managed by LS library |
| Flash | OK | Managed by the browser |
| Flex | OK | Managed by the browser |
| iOS | OK | It's the default for apps |
| Java ME | TO BE ENFORCED | Just issue: java.net.CookieHandler.setDefault(new java.net.CookieManag |

| | | |
|---|---|---|
| | | er(null, java.net.CookiePolicy.ACCEPT_ALL)); |
| Blackberry | TO BE ENFORCED | Just issue: java.net.CookieHandler.setDefault(new java.net.CookieManager(null, java.net.CookiePolicy.ACCEPT_ALL)); |

**WebSocket**: WebSocket is not currently used by non-browser clients, which rely exclusively on HTTP(S).

## 2.2 Leverage Application Stickiness – Options 1.B

With these options, stickiness is provided by application-level mechanisms, rather than leveraging the stickiness mechanisms offered by the Load Balancer.

As we will show, this involves the setting of the `<control_link_address>` and/or `<control_link_machine_name>` Server configuration elements.

For browser-based clients and only for startup performance reasons, the following should also be added to the client initialization code, before connecting:

```
lsClient.connectionOptions.setEarlyWSOpenEnabled(false);
```

### 2.2.1 Option 1.B.1 – Servers behind the Load Balancer

The final nodes can be directly addressed by the Client through the hostnames, which the Load Balancer uses to determine the Lightstreamer Server box to forward the request to.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 80 |
| CR | push1.mycompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 1 | 10.0.0.1 | 80 |
| CR | push2.mycompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 2 | 10.0.0.2 | 80 |

The first column of the table above shows the types of connection that will reach the target box.
The two instances of the Lightstreamer Server do not need to have any public IP addresses, because VIPs are used in the Load Balancer.
All of the requests will reach the Load Balancer, which must adopt a routing algorithm based on the hostname with which the request is received (L7 or HTTP switching). If the host name is “push”, it will use the balancing algorithm (e.g. round robin). If the host name is “push1”, it will send the request to Server 1. If the host name is “push2”, it will send the request to Server 2.

The lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
<control_link_address>push2.mycompany.com</control_link_address>
```

Notice that the same solution as Option 1.B.2 could be used too (use three different VIPs on the Load Balancer rather than doing L7 switching).

**WebSocket**: If the Balancer supports WebSocket, the connections will be automatically upgraded from HTTP when necessary, otherwise the Client will detect WebSocket failure and will keep using HTTP.

#### 2.2.1.1 Example of Multihosting

Only in case of a browser-based client in which the document.domain property is set in the page, the requirement that the Server hostname must be consistent with the domain set causes the previous configuration not to support a multihosting scenario, where the same Lightstreamer Servers are used to serve clients connected to different domains (e.g. “mycompany.com” and “ourcompany.com”).

In order to cope with this case, the `<control_link_machine_name>` setting can be used in addition to the `<control_link_address>` setting. This allows the Server to identify itself in different ways, according to the domain setting of the requesting page.
Note that the `<control_link_machine_name>` setting does not replace the `<control_link_address>` setting; the former will only override the latter when the browser domain check is involved; in all other cases, the `<control_link_address>` will still be needed.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 80 |
| CR | push1.mycompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 1 | 10.0.0.1 | 80 |
| CR | push2.mycompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 2 | 10.0.0.2 | 80 |
| S | push.ourcompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 80 |
| CR | push1.ourcompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 1 | 10.0.0.1 | 80 |
| CR | push2.ourcompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 2 | 10.0.0.2 | 80 |

The Lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
<control_link_machine_name>push1</control_link_machine_name>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
<control_link_machine_name>push2</control_link_machine_name>
<control_link_address>push2.mycompany.com</control_link_address>
```

Similar considerations apply to all the other cases in which the `<control_link_address>` setting is levaraged.

### 2.2.2 Option 1.B.2 – Servers partially behind the Load Balancer

The final nodes can be directly accessed through public IP addresses assigned to the Lightstreamer Server boxes.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer | 200.0.0.10 | 80 | LS Server 1 or LS Server 2 (balancing) | 200.0.0.1 or 200.0.0.2 | 80 |
| CR | push1.mycompany.com | LS Server1 | 200.0.0.1 | 80 | - | - | - |
| CR | push2.mycompany.com | LS Server2 | 200.0.0.2 | 80 | - | - | - |

The two instances of the Lightstreamer Server should be directly reachable from the Internet (i.e. each of them has a public IP address), besides being reachable through the Load Balancer.

The lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
<control_link_address>push2.mycompany.com</control_link_address>
```

**WebSocket**: This option is particularly useful If the Balancer does not support WebSocket. Only the “S” connection (which is always HTTP-based) passes through the Balancer, while the “CR” connections (which can be upgraded to WS) go straight to the Lightstreamer Servers.

# 3 HTTPS-based Scenarios

HTTPS has a limitation with respect to HTTP. Since the TLS/SSL handshake is usually performed before the application protocol (HTTP) can start[1], the Balancer cannot detect the hostname indicated in the application request, because it is encrypted and only the final Server can decode it (unless SSL Acceleration is used), so, for example, Option 1.B.2 is not possible. Furthermore, the same instance of the Lightstreamer Server should be accessed with two different hostnames; but a traditional SSL certificate associated to the server socket can contain only one hostname, resulting in a security warning on the Client when the second name is used. Two solutions exist to this issue: using Wildcard SSL Certificates, or using multiple certificates on different sockets.

Note that some options below use an **SSL Acceleration** (a.k.a. **SSL offloading**) module, a hardware component that off-loads the encryption/decryption operations from the Lightstreamer Server and then uses plain HTTP to communicate with the Lightstreamer Server. Please consider that sometimes SSL offload/acceleration products introduce proxy connections at some level. This could have a double impact on Lightstreamer's features: 1) If the SSL module buffers the full response, before forwarding it to the client, the client will automatically switch from streaming mode to polling mode. 2) The connection that the Lightstreamer Server monitors is not the actual client connection, but is the connection with the SSL Accelerator; if the Accelerator introduces a TCP buffer, the adaptive streaming algorithms of Lightstreamer could take a longer time to detect a network congestion.

## 3.1 Leverage LB Stickiness – Options 2.A

The "**sticky**" property of the Load Balancer is leveraged. Usually Load Balancers offer at least two types of stickiness mechanisms, which leads to two distinct options.

### 3.1.1 Option 2.A.1 – IP-address based

The Balancer routes the requests based on the source IP address. After a request from a new IP address is received, all the subsequent requests from the same IP address are routed to the same node chosen for the first request. The same considerations as in Option 1.A.1 apply from a theoretical point of view (potentially making this option unreliable), but client IP switching does not usually happen with HTTPS.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S, CR | push.mycompany.com | Load Balancer | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing + sticky) | 10.0.0.1 or 10.0.0.2 | 443 |

The lightstreamer_conf.xml file will be the same for both the Servers and will contain the following element:

```
<https_server name="HTTPS Server">
   <port>443</port>
   <keystore>
```

[1] See:
- http://en.wikipedia.org/wiki/Secure_Sockets_Layer#Support_for_name-based_virtual_servers
- https://www.switch.ch/pki/meetings/2007-01/namebased_ssl_virtualhosts.pdf

```
      <keystore_file>push.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
```

No clustering configuration through the `<control_link_address>` element is necessary.

**WebSocket**: If the Balancer does not support WebSocket, it is possible to configure it at level-4 (TCP), rather than at level-7 (HTTP).

#### 3.1.1.1 Example of Multihosting

Only in case of a browser-based client in which the document.domain property is set in the page, the requirement that the Server hostname must be consistent with the domain set causes the previous configuration not to support a multihosting scenario, where the same Lightstreamer Servers are used to serve clients connected to different domains (e.g. “mycompany.com” and “ourcompany.com”).

In order to cope with this case, we must extend the configuration in order to allow each Server instance to provide different SSL certificates based on the hostname with which it is accessed.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S, CR | push.mycompany.com | Load Balancer | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing + sticky) | 10.0.0.1 or 10.0.0.2 | 443 |
| S, CR | push.ourcompany.com | Load Balancer | 200.0.0.20 | 443 | LS Server 1 or LS Server 2 (balancing + sticky) | 10.0.0.1 or 10.0.0.2 | 444 |

Two traditional SSL certificates are needed for the two different hostnames.

The Lightstreamer_conf.xml file should contain the following elements, for both the Lightstreamer Server 1 and Lightstreamer Server 2:

```
<https_server name="My HTTPS Server">
   <port>443</port>
   <keystore>
      <keystore_file>mypush.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<https_server name="Our HTTPS Server">
   <port>444</port>
   <keystore>
      <keystore_file>ourpush.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
```

Similar considerations apply to all the other cases in which individual SSL certificates have to be provided directly by Lightstreamer Server.

As an alternative, a Multi-Domain SSL Certificate could be used, so that one certificate would be enough for all the Lightstreamer Server instances. But please consider that Multi-Domain Certificates have not been tested with Lightstreamer yet, though they should work[2].

### 3.1.2 Option 2.A.2 – Cookie based

The Balancer must support a "Sticky Cookie-Insert" mode, provided that the Load Balancer is extended with an **SSL Acceleration** module (otherwise the encrypted cookies would not be writeable and readable by the Balancer). The same considerations as in Option 1.A.2 apply.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S, CR | push.mycompany.com | Load Balancer + SSL Accel. | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing + sticky) | 10.0.0.1 or 10.0.0.2 | 80 |

The lightstreamer_conf.xml file will be the same for both the Servers and will contain the following element:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
```

No clustering configuration through the `<control_link_address>` element is necessary.

For non-browser clients, see the additional notes provided in section 2.1.2.1.

**WebSocket**: As this option works at level 7 (level 4 is impossible because it does not allow cookie insertion), it is required that the Balancer supports WebSocket.

## 3.2 Leverage Application Stickiness – Options 2.B

With these options, stickiness is provided by application-level mechanisms, rather than leveraging the stickiness mechanisms offered by the Load Balancer.

As we will show, this involves the setting of the `<control_link_address>` and/or `<control_link_machine_name>` Server configuration elements.

For browser-based clients and only for startup performance reasons, the following should also be added to the client initialization code, before connecting:

```
lsClient.connectionOptions.setEarlyWSOpenEnabled(false);
```

[2] For more information, see for example:
- http://www.whichssl.com/mdc.html
- http://www.jguru.com/forums/view.jsp?EID=1358409

### 3.2.1 Leverage Wildcard Certificates – Options 2.B.1

A Wildcard SSL Certificate helps enable SSL encryption on multiple sub-domains using a single certificate as long as the domains are controlled by the same organization and share the same second-level domain name. For example, a certificate issued using the Common Name "***.mycompany.com**" may be used to secure all the following domains: "push.mycompany.com", "push1.mycompany.com", "push2.mycompany.com".

The use of Wildcard Certificates is addressed in RFC 2818[3], section 10. Browser compatibility is quite high, where typically only older micro-browsers do not support wildcard certificates[4]. For more information on wildcard certificates, including possible limitation, you may visit the specific pages of some well-known certification authorities sites[5].

#### 3.2.1.1 Option 2.B.1.1 – Servers behind the Load Balancer

An SSL certificate is issued using the Common Name "***.mycompany.com**".

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 443 |
| CR | push1.mycompany.com | Load Balancer | 200.0.0.1 | 443 | LS Server 1 | 10.0.0.1 | 443 |
| CR | push2.mycompany.com | Load Balancer | 200.0.0.2 | 443 | LS Server 2 | 10.0.0.2 | 443 |

The Load Balancer needs to be exposed on multiple IP addresses (in this example three) in order to distinguish the final destination of the request, because it cannot read the encrypted hostname that is part of the HTTPS request (however it is necessary to use multiple hostnames, because the Web Client always needs to access the Server with a logical name and not an IP address, due to security restraints).

The lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<https_server name="HTTPS Server">
   <port>443</port>
   <keystore>
      <keystore_file>wildcard_push.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<https_server name="HTTPS Server">
   <port>443</port>
```

[3] http://www.ietf.org/rfc/rfc2818.txt

[4] For example, see http://www.geocerts.com/ssl/browsers

[5] For example, see:
- http://www.verisign.com/ssl-certificates/wildcard-ssl-certificates/
- https://www.thawte.com/ssl-digital-certificates/wildcardssl/index.html

```
    <keystore>
       <keystore_file>wildcard_push.keystore</keystore_file>
       <keystore_password>mypassword</keystore_password>
    </keystore>
</https_server>
<control_link_address>push2.mycompany.com</control_link_address>
```

**WebSocket**: As this option works at level 4 (level 7 is impossible because SSL offloading is not being used), WebSocket should always work with no issues.

#### 3.2.1.1.1 Example of Multihosting

As already pointed out in section 3.1.1.1, only in case of a browser-based client in which the document.domain property is set in the page, the requirement that the Server hostname must be consistent with the domain set causes the previous configuration not to support a multihosting scenario.
In this example, Option 2.B.1.1 is used to implement a such a scenario, where the same Lightstreamer Servers are used to serve clients connected to different domains (e.g. “mycompany.com” and “ourcompany.com”).

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 443 |
| CR | push1.mycompany.com | Load Balancer | 200.0.0.1 | 443 | LS Server 1 | 10.0.0.1 | 443 |
| CR | push2.mycompany.com | Load Balancer | 200.0.0.2 | 443 | LS Server 2 | 10.0.0.2 | 443 |
| S | push.ourcompany.com | Load Balancer | 200.0.0.20 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 444 |
| CR | push1.ourcompany.com | Load Balancer | 200.0.0.11 | 443 | LS Server 1 | 10.0.0.1 | 444 |
| CR | push2.ourcompany.com | Load Balancer | 200.0.0.12 | 443 | LS Server 2 | 10.0.0.2 | 444 |

Two wildcard SSL certificates are needed for the two different domains.

The Lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<https_server name="My HTTPS Server">
   <port>443</port>
   <keystore>
      <keystore_file>wildcard_mypush.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<https_server name="Our HTTPS Server">
   <port>444</port>
   <keystore>
      <keystore_file>wildcard_ourpush.keystore</keystore_file>
```

```
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_machine_name>push1</control_link_machine_name>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<https_server name="My HTTPS Server">
   <port>443</port>
   <keystore>
      <keystore_file>wildcard_mypush.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<https_server name="Our HTTPS Server">
   <port>444</port>
   <keystore>
      <keystore_file>wildcard_ourpush.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_machine_name>push2</control_link_machine_name>
<control_link_address>push2.mycompany.com</control_link_address>
```

As an alternative, the same considerations on Multi-Domain SSL Certificate reported in section 3.1.1.1 apply.

#### 3.2.1.2 Option 2.B.1.2 – Servers partially behind the Load Balancer

An SSL certificate is issued using the Common Name "***.mycompany.com**".
The final nodes can be directly accessed through public IP addresses assigned to the Lightstreamer Server boxes.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing) | 200.0.0.1 or 200.0.0.2 | 443 |
| CR | push1.mycompany.com | LS Server1 | 200.0.0.1 | 443 | - | - | - |
| CR | push2.mycompany.com | LS Server2 | 200.0.0.2 | 443 | - | - | - |

The two instances of the Lightstreamer Server should be directly reachable from the Internet (i.e. each of them has a public IP address), besides being reachable through the Load Balancer.

The lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<https_server name="HTTPS Server">
   <port>443</port>
   <keystore>
      <keystore_file>wildcard_push.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<https_server name="HTTPS Server">
   <port>443</port>
   <keystore>
      <keystore_file>wildcard_push.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_address>push2.mycompany.com</control_link_address>
```

**WebSocket**: Only the “S” connection (which is always HTTPS-based) passes through the Balancer, while the “CR” connections (which can be upgraded to WSS) go straight to the Lightstreamer Servers. Furthermore, this option works at level 4 (level 7 is impossible because SSL offloading is not being used). So, WebSocket should always work with no issues.

#### 3.2.1.3 Option 2.B.1.3 – Servers behind an SSL Accelerator

An SSL certificate is issued using the Common Name “***.mycompany.com**”.
An external SSL Accelerator is used to offload the SSL operations, though the limitations outlined at the beginning of section 3 should be taken into consideration.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer + SSL Accel. | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 80 |
| CR | push1.mycompany.com | Load Balancer + SSL Accel. | 200.0.0.10 | 443 | LS Server 1 | 10.0.0.1 | 80 |
| CR | push2.mycompany.com | Load Balancer + SSL Accel. | 200.0.0.10 | 443 | LS Server 2 | 10.0.0.2 | 80 |

The Lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
<control_link_address>push2.mycompany.com</control_link_address>
```

**WebSocket**: If the Balancer does not support WebSocket, configuring it at level-4 (TCP) can in some cases make WebSocket work, provided that the Balancer supports level-4 SSL offloading.

### 3.2.1.4 Option 2.B.1.4 – Servers partially behind an SSL Accelerator

An SSL certificate is issued using the Common Name "***.mycompany.com**".
The final nodes can be directly accessed through public IP addresses assigned to the Lightstreamer Server boxes.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer + SSL Accel. | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 80 |
| CR | push1.mycompany.com | LS Server 1 | 200.0.0.1 | 443 | - | - | - |
| CR | push2.mycompany.com | LS Server 2 | 200.0.0.2 | 443 | - | - | - |

The two instances of the Lightstreamer Server listen to HTTP requests on a private-IP socket AND to HTTPS requests on a public-IP socket. The Load Balancer mounts the SSL certificate associated to "push.mycompany.com". The Balancer balances the incoming "S" connections, while the "CR" connections are directly sent to the final Lightstreamer Servers.

The Lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<http_server name="HTTP Server S">
   <port>80</port>
   <listening_interface>10.0.0.1</listening_interface>
</http_server>
<https_server name="HTTPS Server CR">
   <port>443</port>
   <listening_interface>200.0.0.1</listening_interface>
    <keystore>
      <keystore_file>wildcard_push.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<http_server name="HTTP Server S">
   <port>80</port>
   <listening_interface>10.0.0.2</listening_interface>
</http_server>
<https_server name="HTTPS Server CR">
   <port>443</port>
   <listening_interface>200.0.0.2</listening_interface>
    <keystore>
      <keystore_file>wildcard_push.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_address>push2.mycompany.com</control_link_address>
```

**WebSocket**: This option is particularly useful If the Balancer does not support WebSocket. Only the "S" connection (which is always HTTPS-based) passes through the Balancer, while the "CR" connections (which can be upgraded to WSS) go straight to the Lightstreamer Servers.

### 3.2.2 Leverage Individual Certificates – Options 2.B.2

In case wildcard certificates cannot be used, individual certificates should be issued for each hostname used in the cluster.

#### 3.2.2.1 Option 2.B.2.1 – Servers behind the Load Balancer

Two different SSL certificates (for two different hostnames) need to be handled by each node of the Lightstreamer Server, which needs to create two server sockets associated to two different keystores. For example, the server socket on port 443 will handle the “push.mycompany.com” certificate, while the server socket on port 444 will handle the “push1.mycompany.com” certificate. A total of three certificates are needed in this example.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 443 |
| CR | push1.mycompany.com | Load Balancer | 200.0.0.1 | 443 | LS Server 1 | 10.0.0.1 | 444 |
| CR | push2.mycompany.com | Load Balancer | 200.0.0.2 | 443 | LS Server 2 | 10.0.0.2 | 444 |

The Load Balancer needs to be exposed on multiple IP addresses (in this example three) in order to distinguish the final destination of the request, because it cannot read the encrypted hostname that is part of the HTTPS request (however it is necessary to use multiple hostnames, because the Web Client always needs to access the Server with a logical name and not an IP address, due to security restraints).

The lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<https_server name="HTTPS Server S">
   <port>443</port>
   <keystore>
      <keystore_file>push.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<https_server name="HTTPS Server CR">
   <port>444</port>
    <keystore>
      <keystore_file>push1.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<https_server name="HTTPS Server S">
   <port>443</port>
   <keystore>
      <keystore_file>push.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
```

```
   </keystore>
</https_server>
<https_server name="HTTPS Server CR">
   <port>444</port>
    <keystore>
      <keystore_file>push2.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_address>push2.mycompany.com</control_link_address>
```

**WebSocket**: As this option works at level 4 (level 7 is impossible because SSL offloading is not being used), WebSocket should always work with no issues.

#### 3.2.2.1.1 Example of Multihosting

As already pointed out in sections 3.1.1.1 and 2.2.1.1, only in case of a browser-based client in which the document.domain property is set in the page, the requirement that the Server hostname must be consistent with the domain set causes the previous configuration not to support a multihosting scenario.
In this example, Option 2.B.2.1 is used to implement a such a scenario, where the same Lightstreamer Servers are used to serve clients connected to different domains (e.g. “mycompany.com” and “ourcompany.com”).

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 443 |
| CR | push1.mycompany.com | Load Balancer | 200.0.0.1 | 443 | LS Server 1 | 10.0.0.1 | 444 |
| CR | push2.mycompany.com | Load Balancer | 200.0.0.2 | 443 | LS Server 2 | 10.0.0.2 | 444 |
| S | push.ourcompany.com | Load Balancer | 200.0.0.20 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.11 or 10.0.0.12 | 443 |
| CR | push1.ourcompany.com | Load Balancer | 200.0.0.11 | 443 | LS Server 1 | 10.0.0.11 | 444 |
| CR | push2.ourcompany.com | Load Balancer | 200.0.0.12 | 443 | LS Server 2 | 10.0.0.12 | 444 |

Six SSL certificates are needed for the six different hostnames.

The Lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<https_server name="My HTTPS Server S">
   <port>443</port>
   <listening_interface>10.0.0.1</listening_interface>
   <keystore>
      <keystore_file>mypush.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
```

```
      </keystore>
   </https_server>
   <https_server name="My HTTPS Server CR">
      <port>444</port>
      <listening_interface>10.0.0.1</listening_interface>
       <keystore>
         <keystore_file>mypush1.keystore</keystore_file>
         <keystore_password>mypassword</keystore_password>
      </keystore>
   </https_server>
   <https_server name="Our HTTPS Server S">
      <port>443</port>
      <listening_interface>10.0.0.11</listening_interface>
      <keystore>
         <keystore_file>ourpush.keystore</keystore_file>
         <keystore_password>mypassword</keystore_password>
      </keystore>
   </https_server>
   <https_server name="Our HTTPS Server CR">
      <port>444</port>
      <listening_interface>10.0.0.11</listening_interface>
       <keystore>
         <keystore_file>ourpush1.keystore</keystore_file>
         <keystore_password>mypassword</keystore_password>
      </keystore>
   </https_server>
   <control_link_machine_name>push1</control_link_machine_name>
   <control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
   <https_server name="My HTTPS Server S">
      <port>443</port>
      <listening_interface>10.0.0.2</listening_interface>
      <keystore>
         <keystore_file>mypush.keystore</keystore_file>
         <keystore_password>mypassword</keystore_password>
      </keystore>
   </https_server>
   <https_server name="My HTTPS Server CR">
      <port>444</port>
      <listening_interface>10.0.0.2</listening_interface>
       <keystore>
         <keystore_file>mypush2.keystore</keystore_file>
         <keystore_password>mypassword</keystore_password>
      </keystore>
   </https_server>
   <https_server name="Our HTTPS Server S">
      <port>443</port>
      <listening_interface>10.0.0.12</listening_interface>
      <keystore>
         <keystore_file>ourpush.keystore</keystore_file>
         <keystore_password>mypassword</keystore_password>
      </keystore>
   </https_server>
   <https_server name="Our HTTPS Server CR">
      <port>444</port>
      <listening_interface>10.0.0.12</listening_interface>
```

```
    <keystore>
      <keystore_file>ourpush2.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
    </keystore>
</https_server>
<control_link_machine_name>push2</control_link_machine_name>
<control_link_address>push2.mycompany.com</control_link_address>
```

### 3.2.2.2 Option 2.B.2.2 – Servers partially behind the Load Balancer

The two server sockets of the Lightstreamer Server, associated to two different keystores, can be bound to the same ports of two different IP addresses. For example, the server socket on 10.0.0.1:443 (private IP address) will handle the “push.mycompany.com” certificate, while the server socket on 200.0.0.1:443 (public IP address) will handle the “push1.mycompany.com” certificate. This option is certainly less common and less convenient than Option 2.B.2.1, though technically doable.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 443 |
| CR | push1.mycompany.com | LS Server1 | 200.0.0.1 | 443 | - | - | - |
| CR | push2.mycompany.com | LS Server2 | 200.0.0.2 | 443 | - | - | - |

The lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<https_server name="HTTPS Server S">
   <port>443</port>
   <listening_interface>10.0.0.1</listening_interface>
   <keystore>
      <keystore_file>push.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<https_server name="HTTPS Server CR">
   <port>443</port>
   <listening_interface>200.0.0.1</listening_interface>
    <keystore>
      <keystore_file>push1.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<https_server name="HTTPS Server S">
   <port>443</port>
   <listening_interface>10.0.0.2</listening_interface>
   <keystore>
      <keystore_file>push.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<https_server name="HTTPS Server CR">
```

```
    <port>443</port>
    <listening_interface>200.0.0.2</listening_interface>
     <keystore>
       <keystore_file>push2.keystore</keystore_file>
       <keystore_password>mypassword</keystore_password>
    </keystore>
</https_server>
<control_link_address>push2.mycompany.com</control_link_address>
```

**WebSocket**: Only the "S" connection (which is always HTTPS-based) passes through the Balancer, while the "CR" connections (which can be upgraded to WSS) go straight to the Lightstreamer Servers. Furthermore, this option works at level 4 (level 7 is impossible because SSL offloading is not being used). So, WebSocket should always work with no issues.

#### 3.2.2.3 Option 2.B.2.3 – Servers behind an SSL Accelerator

An external SSL Accelerator can be used to offload the SSL operations, though the limitations outlined at the beginning of section 3 should be taken into consideration.

The Load Balancer is extended with an **SSL Acceleration** module, that takes care of the encryption/decryption operations and uses HTTP to communicate with the final cluster nodes. In order for this option to work, the SSL Accelerator must be able to manage different SSL certificates based on the IP address on which it is reached.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer + SSL Accel. | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 80 |
| CR | push1.mycompany.com | Load Balancer + SSL Accel. | 200.0.0.1 | 443 | LS Server 1 | 10.0.0.1 | 80 |
| CR | push2.mycompany.com | Load Balancer + SSL Accel. | 200.0.0.2 | 443 | LS Server 2 | 10.0.0.2 | 80 |

The two instances of the Lightstreamer Server do not need to have any public IP addresses and they are configured to listen only on an HTTP socket. The Load Balancer is exposed to the Clients through three different IP addresses. The Balancer mounts three SSL certificates (associated to the three different hostnames) and chooses the certificate to use based on the target IP address of the Client request. So, all of the requests will reach the Load Balancer, which must adopt a routing algorithm based on the target IP address or –same result– on the hostname with which the request is received (the hostname is discovered after the request has been decrypted). If the target IP is 200.0.0.10, or the hostname is "push", the Balancer will use the balancing algorithm (e.g. round robin). If the target IP is 200.0.0.1, or the hostname is "push1", it will send the request to Server 1. If the target IP is 200.0.0.2, or the hostname is "push2", it will send the request to Server 2.

The Lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
```

```
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<http_server name="HTTP Server">
   <port>80</port>
</http_server>
<control_link_address>push1.mycompany.com</control_link_address>
```

**WebSocket**: If the Balancer does not support WebSocket, configuring it at level-4 (TCP) can in some cases make WebSocket work, provided that the Balancer supports level-4 SSL offloading.

#### 3.2.2.4 Option 2.B.2.4 – Servers partially behind an SSL Accelerator

The final nodes can be directly accessed through public IP addresses assigned to the Lightstreamer Server boxes.

| Conn | Full Hostname | Reached Box | Public IP Address | Public TCP Port | Final Box | Final IP Address | Final Port |
|---|---|---|---|---|---|---|---|
| S | push.mycompany.com | Load Balancer + SSL Accel. | 200.0.0.10 | 443 | LS Server 1 or LS Server 2 (balancing) | 10.0.0.1 or 10.0.0.2 | 80 |
| CR | push1.mycompany.com | LS Server 1 | 200.0.0.1 | 443 | - | - | - |
| CR | push2.mycompany.com | LS Server 2 | 200.0.0.2 | 443 | - | - | - |

The two instances of the Lightstreamer Server listen to HTTP requests on a private-IP socket AND to HTTPS requests on a public-IP socket. The Load Balancer mounts the SSL certificate associated to “push.mycompany.com“. The two Lightstreamer Servers mount the certificates relative to “push1.mycompany.com“ and “push2.mycompany.com“. The Balancer balances the incoming “S” connections, while the “CR” connections are directly sent to the final Lightstreamer Servers.

The Lightstreamer_conf.xml file should contain the following elements:

On Lightstreamer Server 1:

```
<http_server name="HTTP Server S">
   <port>80</port>
   <listening_interface>10.0.0.1</listening_interface>
</http_server>
<https_server name="HTTPS Server CR">
   <port>443</port>
   <listening_interface>200.0.0.1</listening_interface>
    <keystore>
      <keystore_file>push1.keystore</keystore_file>
      <keystore_password>mypassword</keystore_password>
   </keystore>
</https_server>
<control_link_address>push1.mycompany.com</control_link_address>
```

On Lightstreamer Server 2:

```
<http_server name="HTTP Server S">
   <port>80</port>
   <listening_interface>10.0.0.2</listening_interface>
</http_server>
<https_server name="HTTPS Server CR">
   <port>443</port>
```

```
    <listening_interface>200.0.0.2</listening_interface>
     <keystore>
       <keystore_file>push2.keystore</keystore_file>
       <keystore_password>mypassword</keystore_password>
    </keystore>
</https_server>
<control_link_address>push2.mycompany.com</control_link_address>
```

**WebSocket**: This option is particularly useful If the Balancer does not support WebSocket. Only the "S" connection (which is always HTTPS-based) passes through the Balancer, while the "CR" connections (which can be upgraded to WSS) go straight to the Lightstreamer Servers.

# 4 Accessing Server instances directly and the Internal Monitor Console case

In some cases, mainly for testing purpose or for internal use, it may be needed that a browser-based client or an application client connects to a specific instance of the Server in the cluster, by addressing it through a direct hostname.
This is in general possible, but in all scenarios in which the `<control_link_address>` (or `<control_link_machine_name>`) setting is leveraged, that setting would still be used for [**CR**] connections and this could cause the application not to work, in case it were running inside the organization and using a private hostname, whereas the public hostname specified in `<control_link_address>` were not visible.
This limitation can be overcome by instructing the application to ignore the `<control_link_address>` setting. This is currently possible with some client libraries, specifically the JavaScript, Java SE and Android libraries, in which a proper configuration setting has been made available to application code.
For browser-based clients, the following should be added to initialization code before connecting:

```
lsClient.connectionOptions.setServerInstanceAddressIgnored(true);
```

Obviously, a client that ignores the `<control_link_address>` may not work properly if it connects to the Server through the Load Balancer address.

A special case is a demo or test web front-end whose pages are directly hosted by Lightstreamer Server through its basic Web Server support. For instance, the demo applications available in Lightstreamer distribution package are deployed in this way.
These front-ends, typically, don't specify a Server hostname, so they access the Server through the same hostname by which their pages have been requested.
These front-ends could be recalled either through the Load Balancer address or through a direct Server instance hostname. In the latter case, the above considerations hold.
By the way, note that, in both cases, if the `<control_link_address>` setting is leveraged, these front-ends may still be subject to using different hostnames to access the Server, so they may incur in the same constraints of any normal browser-based client, hence in the option of setting the document.domain on the pages to enforce interoperability.

The Internal Monitor Console is also a browser-based application whose front-end is hosted (and directly supplied) by Lightstreamer Server.
Moreover, the Monitor Console is also available for production use.
As the purpose of the Monitor Console is that of inspecting single instances, it is configured to ignore any `<control_link_address>` setting; so it can be recalled through any public or private hostname of any Server instance, whereas recalling it through the Load Balancer address is not supported.