lightStreamer

Adapter Remoting Infrastructure

General Overview

Target: Lightstreamer Server v. 7.4 or greater
Last updated: 8/3/2023

Table of contents

1 The Architecture

The Lightstreamer **Adapter Remoting Infrastructure** (**ARI**) provides an easy way to develop Remote Adapters for Lightstreamer Server. “Remote” means that an Adapter does not run within the same process as the Java Virtual Machine running the Lightstreamer Server, but within a different process. Such process can be either local (on the same box) or actually remote (on a different box).

So, the advantages of a Remote Adapter with respect to a standard in-process Java Adapter are the following:

- Complete physical decoupling between the code of the Server and the code of the Adapter, avoiding possible interferences (memory usage, thread management, etc.).
- Possibility of deploying the Adapter on a different box and, if necessary, behind a second firewall. Thanks to the support for TLS communication and password-based authentication, the Adapter-server communication could even occur across the outer Internet.
- Possibility of implementing the Adapter in any language, rather than Java. For instance, the provided SDKs for .NET Standard Adapter development, for Node.js adapter development, and for Python Adapter development are actually based on the ARI.

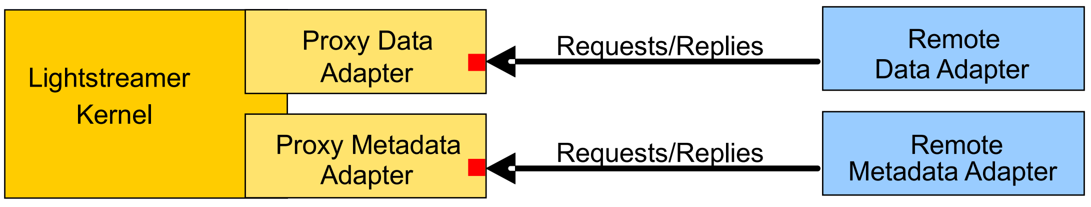
But there is also a drawback. Every interaction between Lightstreamer Server and a Remote Adapter requires information to be exchanged over TCP packets, rather than communicating in-process, resulting in potentially poorer performance. For this reason, some highly intensive callbacks of the Metadata Adapter interface are not supported by the ARI (for example, isSelected()).

Proxy Adapters

The communication between Lightstreamer Server and a Remote Adapter is based on unidirectional channels:

- **requests channel** (data flow from Lightstreamer Server to the Remote Adapter)
- **replies channel** (data flow from the Remote Adapter to Lightstreamer Server)

The figure below shows the general architecture of the ARI:



ARI Protocol

In fact, Lightstreamer Adapter interface mainly consists of method invocations by the Kernel to the Adapters, with related return results or (in the case of the Data Adapter) further update messages.

NOTE: For the sake of clarity, there are also a few cases available of requests originating from the Adapter targeted to the Server. This regards, in particular, the Metadata Adapter interface. In these cases, the requests will flow on the replies channel and the related replies will flow on the requests channel. In practice, the names of the channels are, conventionally, based on the most common pattern of communication.

The protocol used for the communication between the Proxy Adapters and their Remote Adapter counterparts (called ARI protocol) is fully documented as part of the SDK for Generic Adapters.

The **Proxy Data Adapter** and the **Proxy Metadata Adapter** are ready-made Adapters, written in Java and provided as part of the Lightstreamer Server. In practice, they expose the Data Provider and Metadata Provider interfaces over TCP/IP sockets through the ARI protocol.
More precisely, each Proxy Adapter exposes one server socket on which it accepts one connection from the Remote Adapter. This connection will transport the data flows of the two channels.

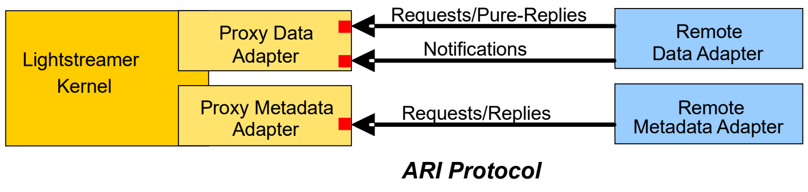
In practice, from a TCP/IP perspective, the Remote Adapters are always clients, that is, they open TCP connections to Lightstreamer Server.
Also with respect to encryption and authentication, when leveraged, the Remote Adapters act as clients: they receive and validate the TLS certificate from the Proxy Adapters and provide their credentials, to be checked by the Proxy Adapters. Moreover, the Proxy Adapter can be configured to request and validate a TLS certificate issued by the Remote Adapter.

Old two-connections case

The **Proxy Data Adapter** also supports two-connections communication. This was the only available case in Server versions earlier than 7.4, hence it is still needed to ensure backward compatibility with old Remote Data Adapters and old Remote Adapter SDKs. Note that the most recent Remote Adapter SDK versions may even no longer support the two-connections behavior.

In this architecture, the replies channel is split into a **pure-replies channel** and a **notifications channel** and the Proxy Data Adapter exposes two server sockets, on which it accepts one connection each from the Remote Data Adapter. Then, the first socket is used to transport the data flow of the **requests** channel and the **pure-replies channel**, whereas the second socket is used unidirectionally, to transport the data flow of the **notifications** channel (from the Remote Data Adapter to the Proxy Data Adapter).
The protocol details are also documented as part of the SDK for Generic Adapters.

The figure below recalls this old architecture:



Remote Servers

There is always a one-to-one relationship between a Proxy Adapter instance and a Remote Adapter instance.
When Lightstreamer Server is launched, with a configured Proxy Adapter, the latter listens on its server socket(s) waiting for Remote Adapters to connect. After a Remote Adapter has connected and,

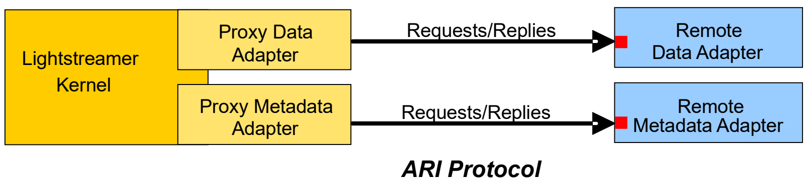
if required, successfully authenticated, its connection will be kept open forever. Multiple concurrent connection attempts on the same server socket are supported, but, as soon as the first one succeeds, the other ones will be truncated.

On the other hand, there is no requirement that each Remote Adapter should live in a separate process.
A process running one or multiple Remote Adapter instances is called Remote Server. The Remote Server executable is a custom component. It can leverage one of the available Remote Adapter SDKs, like the SDK for .NET Standard Adapter development, or only obey the ARI Protocol specifications documented as part of the SDK for Generic Adapters.
If multiple Remote Adapters are needed, how to host them by Remote Servers can be decided freely.

NOTE: Lightstreamer Server performs Adapters initialization in parallel. Hence, if you have set up a Remote Server which manages several Remote Adapters, their connections to the related Proxy Adapters can be performed in sequence and no specific order is required.
Just consider that, after the connection to a Proxy Adapter has succeeded, there is no formal guarantee that the connection to the next Proxy Adapter will succeed immediately; a short time gap may occur on Lightstreamer Server between the opening of the server sockets for different Proxy Adapters.

Connection inversion

As said, from a TCP/IP perspective, the Remote Adapters are always clients, that is, they open TCP connections to Lightstreamer Server. Actually, it is also possible to configure a Proxy Adapter to behave as a TCP client and to open a connection to a listening port on a supplied Remote Server url. Obviously, with this configuration, the Remote Server is required to open a server socket. This is exemplified by the figure below:



Note that the Data Adapter’s two-connection behavior is not supported in this case.

This option is provided only for convenience, but it is not the preferred one for security reasons. In fact, assuming that the Remote Server stays on the back-end and Lightstreamer Server, which accepts client connections, stays in a DMZ, it is more secure, from a firewall perspective, to allow outgoing connections from the back-end rather than incoming ones.

The main advantage of having the Proxy Adapters as TCP clients is that the Remote Servers don’t need to know the addresses of the LS Server instances. This is particularly handy when there is a dynamic cluster of LS Servers, which has to be matched by an equivalent cluster of Remote Adapters.
In this case, when a new LS Server instance is created, the Proxy Adapter will just request the connection to a Load Balancer, which will take care of spawning a new Remote Server instance and connecting to it.
Moreover, the Remote Servers can be made able to accept multiple connections, by spawning a new Remote Adapter for each connection request. In this way, the LS Server instances and Remote Server instances can be completely decoupled: when a new LS Server instance is created and the Proxy Adapter requests the connection to the Load Balancer, the latter can redirect the connection to one of the available Remote Servers.

As for authentication, it is still the Remote Adapter that sends its credentials to be checked by the Proxy Adapter. Moreover, with TLS, the Proxy Adapter also authenticates the Remote Server’s TLS certificate. To have the Remote Server authenticate the Proxy Adapter, the latter can be configured to send its own TLS certificate, which allows the Remote Server to perform the check.

Obviously, leveraging this options adds complexity to the Remote Server, which has to implement a server socket (and, in case of TLS connection, provide a TLS certificate).
Also note that the option is not compatible with all the available Remote Adapter SDKs. In particular, it is not compatible with the use of the current version of the SDK for Python Adapter development, as the connections are handled by the SDK library.

Robust Proxies

There exist two flavors of Proxy Data Adapter: **NetworkedDataProvider** and **RobustNetworkedDataProvider**. Likewise, there exist two flavors of Proxy Metadata Adapter: **NetworkedMetadataProvider** and **RobustNetworkedMetadataProvider**.

Closing the connection could provoke the automatic shutdown of Lightstreamer Server, depending on the type of Proxy Adapter involved. The **NetworkedDataProvider** and **NetworkedMetadataProvider** shut the Server down, if disconnected from the remote counterpart, in order to keep client and server state consistency (they cannot take any recovery action, so the best action is to have the client reconnect to another server instance in the cluster). On the other hand, the **RobustNetworkedDataProvider** and **RobustNetworkedMetadataProvider** contain some recovery capabilities and avoid to terminate the Lightstreamer Server process. Full details on the recovery behavior are available as inline comments within the example “**adapters.xml**” file available in the “**Lightstreamer/docs/remote_adapter_robust_conf_template**” folder.

Connections unresponsive but not closed can be detected and treated as closed by the Proxy Adapter through a configurable timeout on the activity of the sockets. The mechanism is only meaningful if the Remote Servers are configured to send suitable keepalive packets at regular intervals.

2 Deployment of the Proxy Adapters

All the Proxy Adapters of the Adapter Remoting Infrastructure (i.e. Data and Metadata, in the normal and "Robust" version) are pre-deployed inside the Lightstreamer Server. The only necessary action to enable instances of the available adapters is to point them, using their symbolic names, in the "**adapters.xml**" file(s).

In order to deploy an Adapter Set which includes both a Proxy Metadata Adapter and a Proxy Data Adapter, the following steps should be followed:

1) Create a directory within "**Lightstreamer/adapters**". It can be whatever name (for example, "proxy").
2) Depending on the chosen Proxy Adapter type:
    - If deploying Proxy Adapters with "Robust" Proxy Data and Metadata Adapters, copy the "**adapters.xml**" file located under "**Lightstreamer/docs/remote_adapter_robust_conf_template**" to the "proxy" directory.
    - If deploying normal Proxy Adapters, copy the "**adapters.xml**" file located under "**Lightstreamer/docs/remote_adapter_conf_template**" to the "proxy" directory.
3) Edit the copied "**adapters.xml**" file to configure the Proxy Adapters. See the file for inline comments.

The provided "**adapters.xml**" files are templates showing a typical configuration, where both the Adapters (Data and Metadata) are either normal or "Robust" ones. Heterogeneous configurations are possible too. For example:

- A Proxy Data Adapter together with a provided Java Metadata Adapter (such as the LiteralBasedProvider).
- A Proxy Data Adapter together with a custom Java Metadata Adapter.
- A custom Java Data Adapter together with a Proxy Metadata Adapter.
- A Proxy Data Adapter based on "Robust" type together with a Proxy Metadata Adapter of normal type.
- Etc... Any other combination is possible.

If the defined Adapter Set includes multiple Data Adapters, each of them can be either a Proxy Data Adapter or a custom Java Data Adapter.

In all these cases, just extract the relevant **<metadata_provider>** or **<data_provider>** blocks from the "**adapters.xml**" template files and add them in your own adapters.xml.