# TALQ Application Protocol Transaction Management

TALQ Technical Working Group

March 2015

## References


[1] TALQ Specification Overview – TALQ Consortium White Paper (March 2015)
[2] RFC 6202

# 1. INTRODUCTION

The TALQ Specification defines the application protocol between a Central Management System (CMS) and Outdoor Lighting Networks (OLNs) to enable configuration management, control and monitoring of outdoor lighting systems.

This white paper describes the transaction management functionalities supported by the TALQ application protocol and provides examples as well as recommendations on how such functionalities can be used to handle potential errors or failures in the communication between the CMS and the TALQ Bridge. A complete overview of the TALQ Specification is provided in in [1].

# 2. TALQ Client-Server Communications

The TALQ Application protocol follows a client-server architecture, in which the CMS is the Server that hosts resources, and the TALQ Bridge is a Client that updates statuses and attributes in the CMS, receives commands and requests from the CMS and passes them to the OLN for execution of the corresponding actions. In this model, the CMS (Server) may also asynchronously send notifications to the TALQ Bridge (Client).

Three basic message types are defined in the TALQ Application protocol:

- Request: messages sent from the TALQ Bridge to the CMS;
- Response: messages sent from the CMS to the TALQ Bridge as an answer to a Request;
- Notification: asynchronous messages sent from the CMS to the TALQ Bridge.

Specific messages are defined in the context of each TALQ service, but they are derived from the basic request and response types described above. The content of the TALQ application messages are defined as XML complex types that describe attributes and elements of TALQ logical devices and services. The underlying HTTP messaging support layer is responsible for transporting the messages between the application end points (TALQ Bridge and CMS).

## 2.1 Client to Server Communications

TALQ Bridge (Client) to CMS (Server) communication is based on the HTTP request with a POST action. Upon receipt of such a request, the CMS performs the required action and responds immediately with regard to the status of the action or after a timeout. Figure 1 illustrates the normal request-response mechanism using the HTTP POST method. In this example, the TALQ Bridge requests new device configuration information. The CMS response is sent in the

payload of the HTTP layer acknowledgment to the corresponding request. It is important to note that the client expects any request to be persistent when a HTTP OK is received. This doesn't mean the request is completely handled, but the client assumes requests will not be discarded if the recipient fails for some reason.


Figure 1: Example of Client to Server Communication using HTTP request.

## 2.2 Server to Client (Asynchronous) Communications

The HTTP long polling mechanism [2] is used to enable the CMS (Server) to asynchronously send notification messages to the TALQ Bridge (Client), which can be used for configuration updates and control commands. In this way, the TALQ Bridge keeps an open request for information (long poll) with the CMS by sending a getNotifications message as a normal HTTP request.

The CMS uses the open request to send notifications to the TALQ Bridge whenever needed, for instance, when a particular event, status, or timeout occurs. Examples of such notifications are override light control commands and configuration change notifications. After receiving a notification or after a configurable timeout, the TALQ Bridge shall send another long poll request (getNotifications) to the CMS, so that the CMS will almost always have a pending request that it can use to deliver data to the TALQ Bridge asynchronously.

The Server-Client communication with long polling is illustrated in Figure 2. The getNotifications is sent as an HTTP POST to <cmsUri>/notification, where <cmsUri> is the base URI the TALQ Bridge uses to communicate with the CMS. In this example, the CMS asynchronously sends a lighting control override by responding to the getNotifications request with an override notification message and the TALQ Bridge immediately opens a new long poll once the response is received.

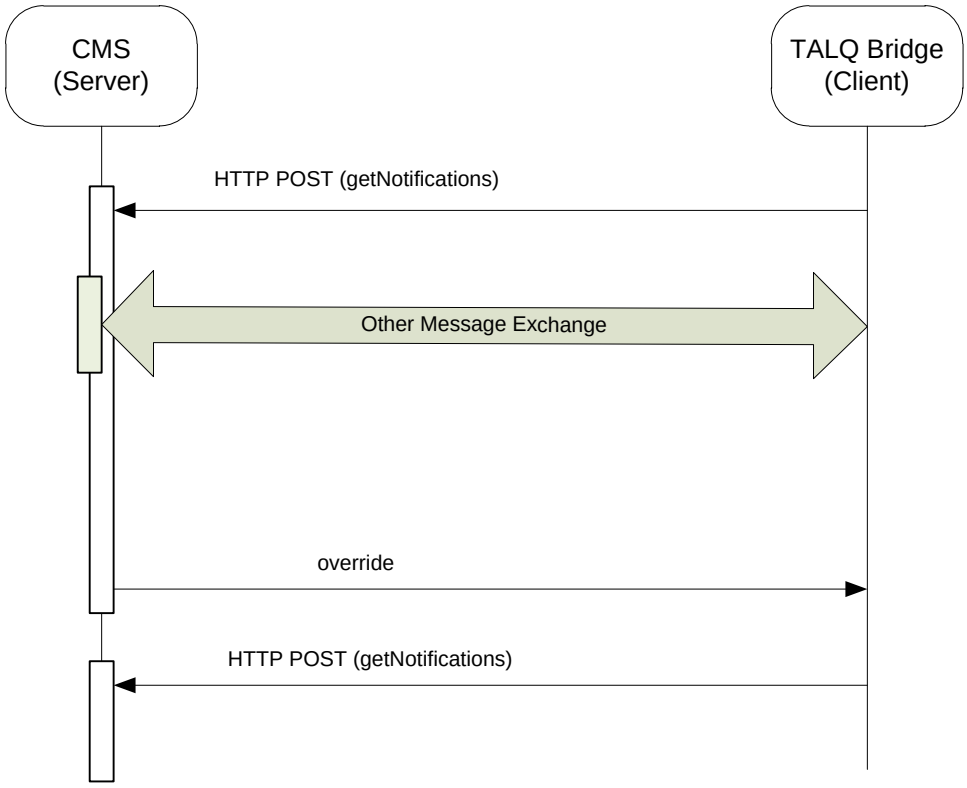


Figure 2: Example of Server-Client communication using HTTP Long Polling.

The TALQ Bridge may also use the HTTP long polling mechanism to control the flow of notifications received from the CMS. For instance, the TALQ Bridge may delay sending new getNotifications messages when it is overloaded handling previously received notifications, or when the load on its network connection with the CMS is high.

As HTTP requests can be terminated due to different reasons, other than a reply from the CMS, the TALQ Bridge is required to at least resend a getNotifications request after a configurable timeout (pollTimeout). Determination of the timeout value is a trade-off between latency in Server-Client message delivery and the processing/network resources needed to transmit additional long poll requests. The timeout values will also depend on specific implementations.

### 2.3 Reliable Data Transfer

Sequence numbers are used in notifications to enable the application end points to handle possible reliability and message transmission errors. Sequence numbers are also used to synchronize TALQ entities at both CMS and TALQ Bridge sides, as well as handle duplicate messages.
Each notification message from the CMS includes a sequence number, which is incremented for every notification or reset to zero when the previous reached the maximum allowed value ($2^{32}$-1).

The TALQ Bridge keeps track of the notification sequence numbers received and handles only notifications received in the correct order. The TALQ Bridge ignores out of order notification messages or stores them for later handling.

The TALQ Bridge also includes a sequence number in the getNotifications requests. In this case, the sequence number is set to the sequence number of the last notification successfully received by the TALQ Bridge. In this way, the CMS can verify which notifications have been successfully received by the TALQ Bridge and optionally retransmit notifications which may have been lost.

The next section describes how the TALQ protocol enables data synchronization and updates using sequence numbers and service specific messages.

## 3. TALQ Data Synchronization

In the TALQ Application Protocol it is essential that both CMS and TALQ Bridge have a consistent view of the logical devices and other entities managed by the TALQ Services, such as calendars, control programs and lamp types. To facilitate data synchronization, the protocol has been designed so that most entities are created and modified (owned) by only one application end point (CMS or TALQ Bridge). The other end point can be informed about any changes on the entity, but only one side can modify it. For instance, calendars and groups are owned by the CMS, whereas logical devices are owned by the TALQ Bridge.

### 3.1 TALQ Entity Management

The TALQ Specification defines a framework to manage entities, where specific messages and content are defined for each type of entity. In general, the message name reflects type of entity.

The following messages are defined to manage entities owned by the CMS:

- <entity>Changed: Notification messages sent from CMS to TALQ Bridge to inform that an entity has changed.
- get<entity>: Request message from TALQ Bridge to CMS to obtain changes for a given entity. This message is triggered by a <entity>Changed notification, if the <entity> configuration is out of date at the TALQ Bridge, or when it discovers an entity not yet known. The TALQ Bridge may also use this message without a preceding <entity>Changed notification. The get<entity> message shall be repeated until an <entity> message is received as an answer. The TALQ Bridge may also use this message to renew the data for an entity if for some reason it determines a synchronization problem with the entity's information. The mechanism for determining a synchronization problem is outside the scope of this specification. Using this message for this specific reason should be performed with care as it might result in excessive data usage on the communication link.
- <entity>: Response message from CMS to the TALQ Bridge containing the requested entity's data.
-

Figure 3 illustrates the generic message exchanges to update an entity (calendar) owned by the CMS. In this example <cmsUri>/control indicates the entity (calendar) being updated is part of the Lighting Control service.

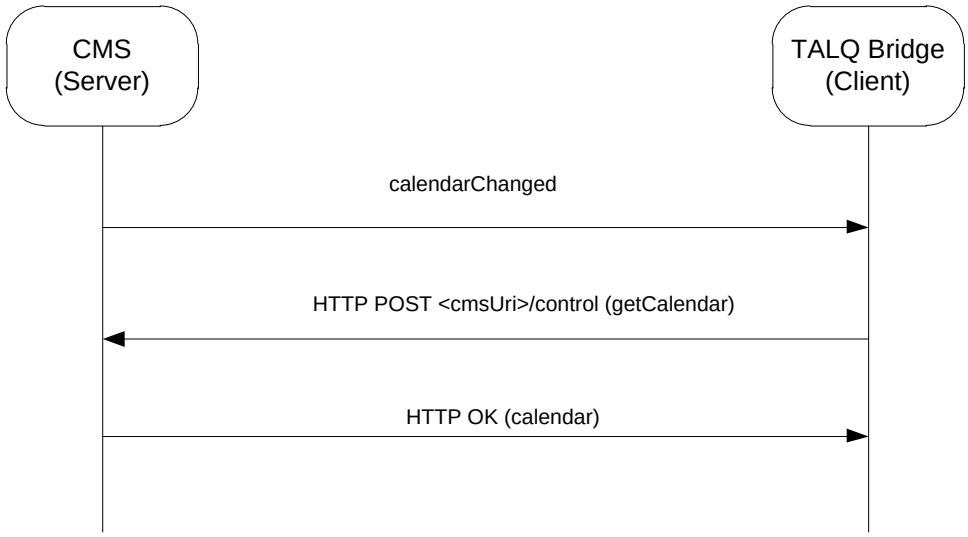


Figure 3: Message exchange to manage an entity (calendar) owned by the CMS.

Similarly, two basic message types are defined to manage entities owned by the TALQ Bridge:

- refresh<Entity>: Notification messages from CMS to TALQ Bridge to ask the TALQ Bridge to retransmit data for a given entity. The CMS uses this message when it discovers an entity not yet known when referred to by another entity. The CMS may also use this message to renew the data for an entity if for some reason it determines a synchronization problem with the entity's information. The mechanism for determining a synchronization problem is outside the scope of this specification.
- update<Entity>: Request messages from TALQ Bridge to CMS to transmit updates for a given entity. The CMS shall acknowledge this request with an HTTP OK. The TALQ Bridge shall repeat the message until it receives a response from the CMS.

Figure 3 illustrates an example of how the TALQ Bridge and the CMS can manage updates on a logical device configuration data, which is owned by the TALQ Bridge. In this example, the CMS asks the TALQ Bridge for a device configuration update. An updateDeviceConfiguration could also be generated by the TALQ Bridge without a refresh notification from the CMS, as it is the owner of the entity and can trigger updates as required.

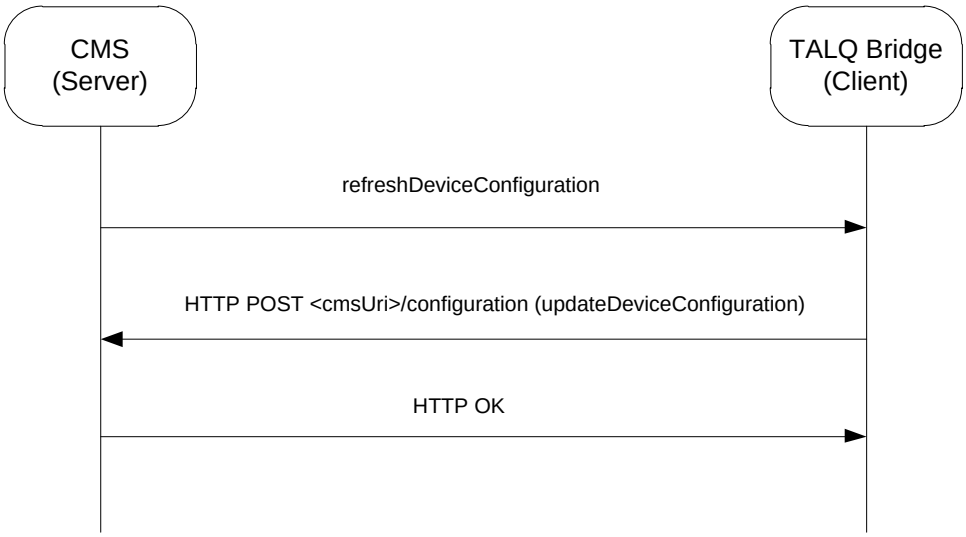


Figure 4: Message exchange to manage an entity (calendar) owned by the TALQ Bridge.

## 3.2 Entity Sequence Numbers

In order to support data synchronization, sequence numbers are also assigned to TALQ entities. The sequence number must be incremented with each update of configuration data associated with the entity. The sequence number is sent together with the data in each

<entity> or update<Entity> message using the 'seq' xml attribute. The receiving side shall maintain a copy of the sequence number and only handle the received <entity> or update<Entity> message when the sequence number in the message is larger than the latest sequence number for the entity. Other messages shall be ignored, although the receiver shall still respond to such ignored messages as if they were handled correctly, otherwise the send may continue to try indefinitely.
In addition, the refresh<Entity> and get<Entity> messages shall contain a 'since' attribute to reflect the current knowledge of the sequence number for an entity at the receiving end. In this way, a sender can transmit only part of the information that changed since the version indicated in the since attribute. The sender may always choose to transmit the complete information. If there are no changes, a message with no changes can be sent with the correct sequence number.

### 3.3 Data Synchronization Examples

This section illustrates how data synchronization can be handled in the TALQ Application protocol by using an example. In this example, the CMS has a calendar update to distribute to luminaire controllers (LC) within an OLN. During the update process, the system must handle communication errors within the OLN.
Figure 5 describes the message exchange sequence in this particular case.

The first point to note is the distinction between the scope of the TALQ Application protocol and the OLN specific protocol between the TALQ Bridge y and the actual luminaire controllers (physical devices).

In the TALQ data model, the physical devices are modeled as three logical devices (LC A, B and C) that are managed by the TALQ Bridge. Furthermore, a calendar is defined/owned by the CMS in order control the behavior of a lighting actuator over time. Once the CMS identified a new calendar update available (for instance, triggered by a user) to be distributed to the OLN, it sends a *calendarChanged* notification to the TALQ Bridge. The TALQ Bridge generates a request for the calendar update (*getCalendar*) including the since attribute with sequence number for the specific calendar maintained by the TALQ Bridge. The CMS then sends the changes in the calendar data according to the since attribute receive in a *calendar* message with the most current sequence number.

Once the calendar message is successfully decoded and handled, it is the responsibility of the TALQ Bridge to distribute the new calendar data to all target devices within the OLN. However, given the device and network conditions within the OLN, the delivery of the calendars to the physical devices may take time. This task should happen in parallel with other transactions being handled. Therefore, the TALQ Bridge sends a new *getNotifications* to keep an open

request for more notifications from the CMS while it tries to deploy the calendar update within the OLN using its own implementation specific protocols. Note that, the TALQ Bridge may decide to delay the notification for other reasons, but it is not required to wait until the calendar data is actually delivered to the devices before sending back a new *getNotifications* request. The next *getNotifications* could even be sent before the *getCalendar* message.

In the next step, the TALQ Bridge sends the update to the target devices using its own OLN_calendar_update message, which could be a simple copy of a TALQ message or a message in vendor specific format, but this is outside the scope of the TALQ Specification. In this example, two updates are successfully delivered (acknowledgements within the OLN, not shown in
Figure 5, could be used for this), but the message to LC C is lost, for instance, due to communication problems with the target device. The TALQ Bridge retries the transmission twice, but after a timeout the data is not yet successfully delivered, and the TALQ Bridge decides to generate an event and report a communication problem with the LC C. The number of retries and timeout value used are implementation specific and depends on the system performance and characteristics. For instance, the TALQ Bridge vendor could decide to trigger a communication failure if the device does not respond within 15 min, an hour or even a day.

The CMS does not expect a specific response from the TALQ to indicate the calendar was successfully deployed. However, it is still important to ensure entities are synchronized on both end points. Reporting the communication failure with the logical device LC C, can help the CMS to identify the device maybe out of sync, but it is still the responsibility of the TALQ Bridge to ensure the logical device is properly updated and configured once the communication is restored. Suppose the failing device was experiencing a power outage in this example. Once, the device comes back to normal, it connects with the TALQ Bridge (OLN_device_update), which can then identify that there is a pending transaction for the device and deliver the calendar update (OLN_calendar_update). The TALQ Bridge also reports the end of the communication failure event by using the startEndFlag in the event log set to false.

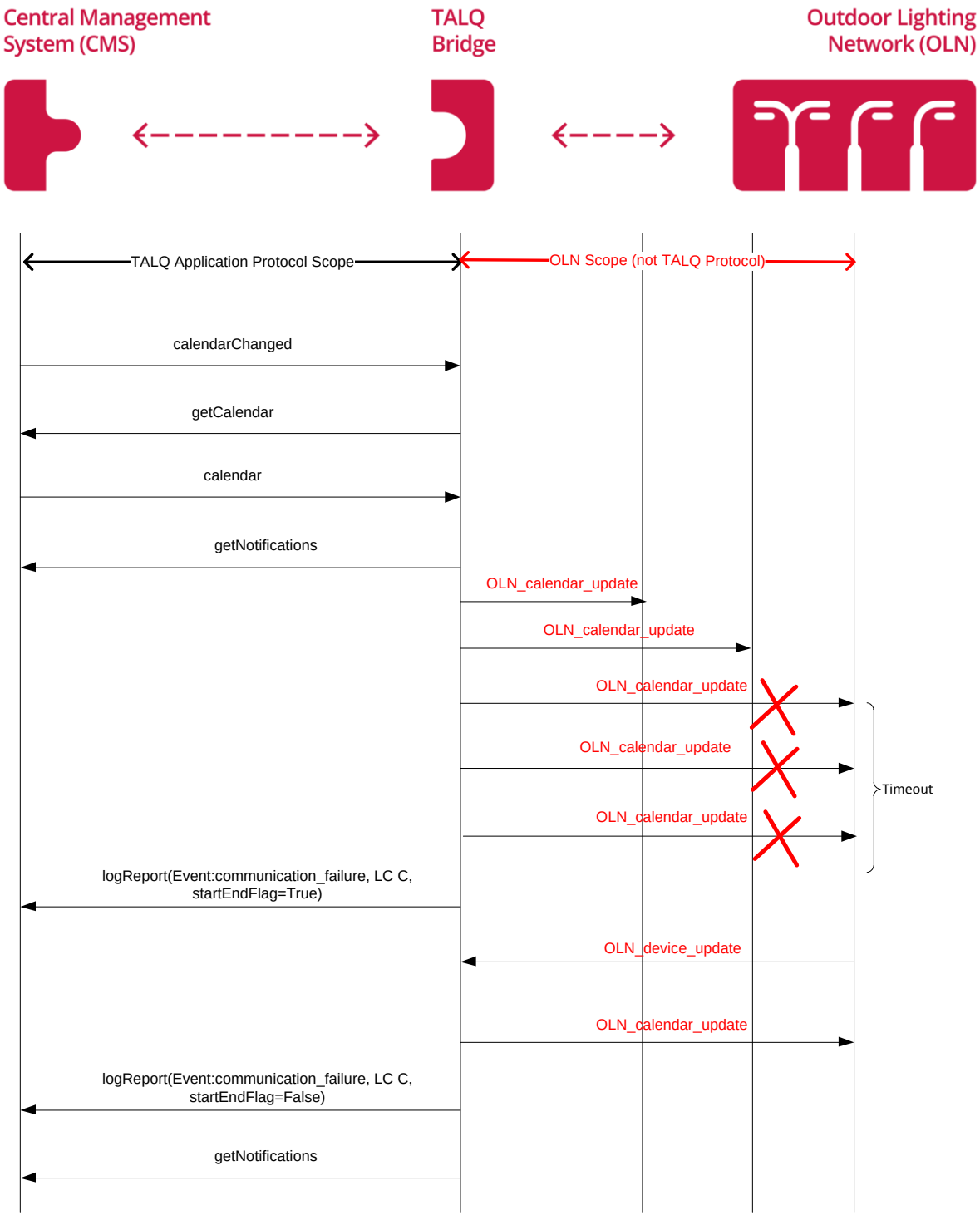


Figure 5:Example of message exchange to manage an entity (calendar).

In another example illustrated in
Figure 6, suppose the CMS and TALQ Bridge are in sync for calendar A. Both have sequence number 5 associated with this calendar and the last notification handled by the TALQ Bridge was 1000.
A user at the CMS changes the calendar A. As a result, the CMS stores the changes and increases the internal calendar sequence number to 6. Next, a *getNotifications* is received with sequence number 1000. The CMS returns a calendarChanged notification with sequence number 1001. The message identifies the change to pertain to calendar A being now at sequence number 6.

After receiving the change notification, the TALQ Bridge compares its current sequence number for calendar A (5) with the new calendar sequence number (6) and decides to retrieve the changes by sending a *getCalendar* request for calendar A with a since attribute set to 5. The CMS returns a calendar message for calendar A with sequence number 6. Finally, the TALQ Bridge stores the changes including the updated sequence number (6) for calendar A and passes it on to the target device (LC A) within the OLN.

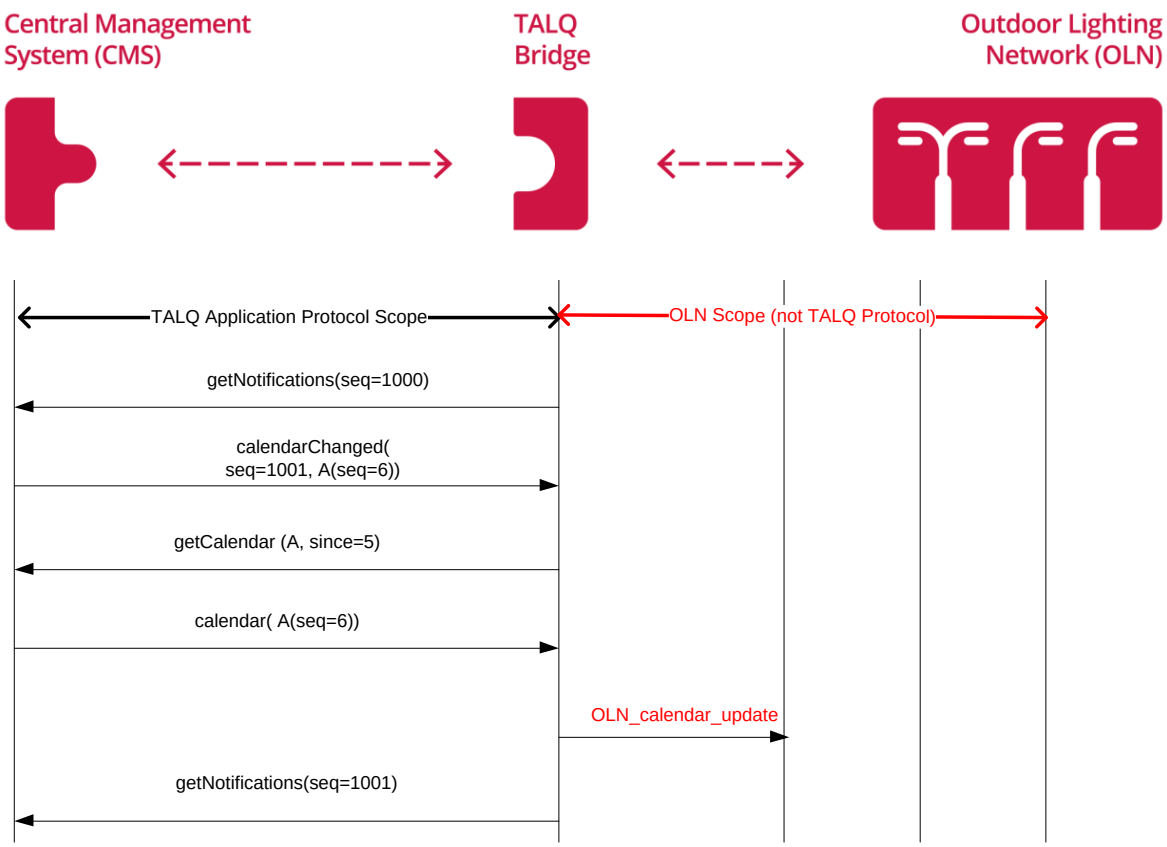


Figure 6: Sequence numbers management example.

## 4. Conclusions

The TALQ Application Protocol defines an extensive and scalable framework transaction management between the CMS and the OLN. The protocol includes the basic features and messages required to ensure data can be correctly synchronized according to the requirements of a typical lighting system implementation. Specific notifications, requests and response messages are defined for every TALQ Service. The protocol is designed with a clear separation of scope between the TALQ and the OLN specific implementations, and as such, it gives full flexibility to vendors to implement and configure their devices according to the needs of the project.