**3GPP TSG-SA5 Meeting #157*****S5-246063***

Hyderabad, India, 14 -18 October 2024 revision of S5-245391

**Source: Ericsson Hungary Ltd.**

**Title: Rel-19 pCR TR 28.871 Reliable notification transfer**

**Document for: Approval**

**Agenda Item: 6.19.8**

# 1 Decision/action requested

***Approve the proposal.***

# 2 References

[1] 3GPP TR 28.871 Study on Service Based Management Architecture enhancement phase 3

[2] 3GPP TS 32.158 Design rules for REpresentational State Transfer (REST) Solution Sets (SS)

# 3 Rationale

Consumers use information received in notifications. Some consumers depend on receiving notifications reliably and in sequence. Notification delivery is not always reliable. Notifications may be missing because the producer was not able to prepare/send them or because they are lost during transfer. Current specifications do not provide mechanisms for detecting missing or out-of-order notifications or a mechanism to recover the missed information.

We propose to improve the reliability of notification handling.

# 4 Detailed proposal

**First change**

## 5.a Reliable notification transfer

### 5.a.1 Description

The handling of notifications has been identified as an area of improvement because there is no guarantee that notifications are always successfully sent to and received by consumers. Consumers might be unaware of the misalignment with the producer. There are multiple conditions that can cause the misalignment:

* Network issues: both temporary and permanent outages. These may be caused both by real connectivity issues and unavailability of the notification recipient.
* Producer internal issues while preparing/sending the notification, (e.g. overload situation, error) may result in notifications not being sent
* Consumer exceptions reported to the producer e.g. with HTTP response codes (4xx, 5xx).
* Notifications may be received out of order e.g. if separate https connections are used to transfer subsequent notifications.
* Consumer issues e.g. abrupt shutdown while processing a notification

We need a reliable mechanism, to detect abnormal conditions and to recover from them. While the heartbeat mechanism checks the network connection, it does not check that the individual notifications are successfully transferred.

To introduce a reliable notification transfer, the following responsibilities have been identified:

* **Producer** should be capable of storing all generated notifications per subscription (actual storage might be optimized to avoid duplicate storage for multiple subscriptions).
* **Producer** should be capable of providing stored notifications, when requested.
* **Producer** should be capable of providing a mechanism to detect missing notifications.
* **Producer** should be capable to detect that a notification is not possible to send out and notify it with a notifyNotificationNotSent message and related state variables
* **Consumer** is responsible for discovering lost or out of order notifications
* **Consumer** should be capable of requesting missing notifications from the producer
* **Consumer** should be capable of processing the notifyNotificationNotSent notification.

The feature should be optional to support, but if supported its use should be mandatory.

### 5.a.2 Potential requirement

**REQ-rel-notif-1:** Producer shall provide a mechanism for the consumer that allows it to discover missing or out-of-order notifications.

**REQ-rel-notif-2:** Producer shall store all generated notifications up to an implementation specific limit.

**REQ-rel-notif-3:** Producer shall provide a mechanism for the consumer that allows it to retrieve the stored notifications from the producer.

**REQ-rel-notif-4:** Producer shall inform the consumer if the producer did not succeed to properly send one or more notifications that should have been sent according to the subscription.

### 5.a.3 Potential solutions

#### 5.a.3.1 Part#1 Add sequence number (REQ-rel-notif-1)

Delivery of notifications can be prevented by many reasons (overload, bugs, firewalls, proxy or concentrator nodes, etc.) some of which cannot be foreseen at the moment. The smartest approach to ensure reliability has been identified as adding a sequence number to all notifications so that the consumer can identify ‘holes’ in the sequence of the notifications and out-of-order notifications. Sequence numbering is a robust mechanism that can detect problems both for the current HTTP 1.1 protocol (see [2]) or any future protocols (e.g. HTTP 3).

Notifications can arrive out-of-order. If the HTTP connection is closed after sending a notification and a new connection is opened for the subsequent notification, the order of delivery is not guaranteed. The sequence number can be used to re-order the notifications.

Every notification should carry a monothonically increasing sequence number, that will be separate for each notification subscription. The sequence number should be a large unsigned integer that is reset to zero at start or restart of the producer.

This will allow the consumer to detect missing or out-of-order notifications. Heartbeat notifications should also carry a new parameter lastEventTime that contains the eventTime parameter from the last (non-heartbeat) notification.

Impacted specifications are: 28.532 and 28.111.

#### 5.a.3.2 Part#2 Retrieve stored notifications (REQ-rel-notif-2 & 3)

To ensure that notifications that failed to be delivered can be retrieved later, for each subscription the producer should store all notifications that were prepared for that specific subscription in a buffer, except notifyHeartbeat messages. The size of the buffer is implementation dependent.

The producer should expose the stored notifications in one or more IOCs per subscription in the NRM (that can be handled by the normal provisioning operations).

Beside exposing the notifications the IOCs may provide

- information about the timespan and/or the sequence number range for which notifications are stored

- a method to retrieve only a specific notification or specific set of notifications

- lastSequenceNumber used (equivalent to the parameter in notifyHeartbeat)

- lastEventTime used (equivalent to the parameter in notifyHeartbeat)

- numberOfStoredNotifications.

Any such attributes should be isNotifiable=False, otherwise an infinite number of value change notifications could be generated.

Other/alternative retrieval methods are for further study.

Impacted specifications are: 28.622 and 28.623.

#### 5.a.3.3 Part#3 Notification could not be sent/prepared (REQ-rel-notif-4)

##### 5.a.3.3.1 Problem Scenario

If a producer knows or suspects that it failed or will fail to prepare or deliver some notifications that are part of a subscription, it should inform the consumer about this.

Situations where notification sending may fail include:

* notifyMOIChanges is not sent if a big set of configuration data is updated in one transaction and the producer doesn’t want to or cannot notify each item
* when due to overload a set of notifications cannot be sent, this can be substituted by sending a single notifyNotificationNotSent
* producer internal SW problems e.g. a SW component restarts in the producer. The notification sender may or may not loose information during this restart

There can be many different situations where a producer is unable to prepare or send out notifications depending on the notification types subscribed but also depending on the implementation of the producer. Due to the many possible failure reasons it would be very difficult for the consumer to monitor all possible underlying reasons e.g. with state variables. To help the consumer (even simple consumers) to detect such producer failures, the subscription should include an overall indication of the problem.

In this problem scenario the original notifications were never prepared/sent. They are not stored, so the sequence number should not be affected.

This function is not an alternative to solutions described in part#1 and part#2. This solution protects against a different problem scenario; thus it should be used together with solutions part#1 and part#2.

##### 5.a.3.3.2 Part#3a notifyNotificationNotSent

In this problem scenario the consumer should send a new notification (type) notifyNotificationNotSent about this problem scenario.

The notification should carry the notificationType(s) of the notifications that were not sent (if it is known). This allows the consumer to limit the recovery action to certain notification types e.g. if only FM is down, there is no need to retrieve CM notifications.

The notifyNotificationNotSent notification should increment and carry the sequence number as all other notifications.

The notification type notifyNotificationNotSent should be implicitly included in all subscriptions, there is no need to include it in the NtfSubscriptionControl.notificationTypes attribute.

Impacted specifications are: 28.622 and 28.623.

##### 5.a.3.3.3 Part#3b state variables on the NtfSubsciptionControl

In this problem scenario the subscription is in a degraded state. This state should be visible not just via the notifyNotificationNotSent notification, it should also be visible by reading NRM attributes. (As a general principle, any data that is available via a notification should also be readable via CRUD operations.)

The producer should provide per subscription

- operationalState attribute

- availabilityStatus attribute potentially with a “dependency/degraded” value

- a list of notification types that are not sent (may be unknown).

##### 5.a.3.3.4 Part#3c Alarm from NtfSubsciptionControl

Implementations may define an alarm to indicate the degraded state of a notification subscription, however a standard alarm is not proposed.

### 5.a.4 Evaluation of potential solutions

Solution part #1, #2 and #3 fulfil the proposed requirements.

Solutions #1, #2 and #3b are recommended for normative work.

It is for further study whether solution #3a should or should not be recommended for normative work.

Solution #3c is not recommended for normative work.

**End of change**