Interface to the Routing System (i2rs)
Internet-Draft
Intended status: Informational
Expires: November 5, 2016

E. Voit
A. Clemm
A. Gonzalez Prieto
Cisco Systems

May 4, 2016

# Requirements for Subscription to YANG Datastores
# draft-ietf-i2rs-pub-sub-requirements-07

Abstract

This document provides requirements for a service that allows client applications to subscribe to updates of a YANG datastore. Based on criteria negotiated as part of a subscription, updates will be pushed to targeted recipients. Such a capability eliminates the need for periodic polling of YANG datastores by applications and fills a functional gap in existing YANG transports (i.e. Netconf and Restconf). Such a service can be summarized as a "pub/sub" service for YANG datastore updates. Beyond a set of basic requirements for the service, various refinements are addressed. These refinements include: periodicity of object updates, filtering out of objects underneath a requested a subtree, and delivery QoS guarantees.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 5, 2016.

Table of Contents

1. Introduction . . . . . 2
2. Business Drivers . . . . . 3
2.1. Pub/Sub in I2RS . . . . . 3
2.2. Pub/Sub variants on Network Elements . . . . . 5
2.3. Existing Generalized Pub/Sub Implementations . . . . . 5
3. Terminology . . . . . 6
4. Requirements . . . . . 7
4.1. Assumptions for Subscriber Behavior . . . . . 7
4.2. Subscription Service Requirements . . . . . 7
4.2.1. General . . . . . 7
4.2.2. Negotiation . . . . . 9
4.2.3. Update Distribution . . . . . 9
4.2.4. Transport . . . . . 10
4.2.5. Security Requirements . . . . . 11
4.2.6. Subscription QoS . . . . . 12
4.2.7. Filtering . . . . . 13
4.2.8. Assurance and Monitoring . . . . . 14
5. Security Considerations . . . . . 14
6. IANA Considerations . . . . . 14
7. Acknowledgements . . . . . 15
8. References . . . . . 15
8.1. Normative References . . . . . 15
8.2. Informative References . . . . . 15
Authors' Addresses . . . . . 16

## 1. Introduction

Applications interacting with YANG datastores require capabilities beyond the traditional client-server configuration of network elements. One class of such applications are service-assurance applications which must maintain a continuous view of operational data and state. Another class of applications are security applications which must continuously track changes made upon network elements to ensure compliance to corporate policy.

Periodic fetching of data is not an adequate solution for applications requiring frequent or prompt updates of remote object state. Applying polling-based solutions here imposes load on

networks, devices, and applications. Additionally, polling solutions are brittle in the face of communication glitches, and have limitations in their ability to synchronize and calibrate retrieval intervals across a network. These limitations can be addressed by including generic object subscription mechanisms within Network Elements, and allowing these mechanisms to be applied in the context of data that is conceptually contained in YANG datastores.

This document aggregates requirements for such subscription from a variety of deployment scenarios.

## 2. Business Drivers

For decades, information delivery of current network state has been accomplished either by fetching from operations interfaces, or via dedicated, customized networking protocols. With the growth of centralized orchestration infrastructures, imperative policy distribution, and YANG's ascent as the dominant data modeling language for use in programmatic interfaces to network elements, this mixture of fetch plus custom networking protocols is no longer sufficient. What is needed is a push mechanism that is able to deliver object changes as they happen.

These push distribution mechanisms will not replace existing networking protocols. Instead they will supplement these protocols, providing different response time, peering, scale, and security characteristics.

Push solutions will not displace all existing operations infrastructure needs. And SNMP and MIBs will remain widely deployed and the defacto choice for many monitoring solutions. But some functions could be displaced. Arguably the biggest shortcoming of SNMP for those applications concerns the need to rely on periodic polling, because it introduces additional load on the network and devices, because it is brittle in case polling cycles are missed, and because is hard to synchronize and calibrate across a network. If applications can only use polling type interaction patterns with YANG datastores, similar issues can be expected.

### 2.1. Pub/Sub in I2RS

Various I2RS documents highlight the need to provide Pub/Sub capabilities between network elements. From [i2rs-arch], there are references throughout the document beginning in section 6.2. Some specific examples include:

o section 7.6 provides high level pub/sub (notification) guidance

- section 6.4.2 identifies "subscribing to an information stream of route changes receiving notifications about peers coming up or going down"

- section 6.3 notes that when local config preempts I2RS, external notification might be necessary

In addition [i2rs-usecase] has relevant requirements. A small subset includes:

- L-Data-REQ-12: The I2RS interface should support user subscriptions to data with the following parameters: push of data synchronously or asynchronously via registered subscriptions...

- L-DATA-REQ-07: The I2RS interface (protocol and IMs) should allow a subscriber to select portions of the data model.

- PI-REQ01: monitor the available routes installed in the RIB of each forwarding device, including near real time notification of route installation and removal.

- BGP-REQ10: I2RS client should be able to instruct the I2RS agent(s) to notify the I2RS client when the BGP processes on an associated routing system observe a route change to a specific set of IP Prefixes and associated prefixes....The I2RS agent should be able to notify the client via publish or subscribe mechanism.

- IGP-REQ-07: The I2RS interface (protocol and IMs) should support a mechanism where the I2RS Clients can subscribe to the I2RS Agent's notification of critical node IGP events.

- MPLS-LDP-REQ-03: The I2RS Agent notifications should allow an I2RS client to subscribe to a stream of state changes regarding the LDP sessions or LDP LSPs from the I2RS Agent.

- L-Data-REQ-01: I2rs must be able to collect large data set from the network with high frequency and resolution with minimal impact to the device's CPU and memory.

And [i2rs-traceability] has Pub/Sub requirements listed in Section 7.4.3.

- I2RS Agents should support publishing I2RS trace log information to that feed as described in [i2rs-arch]. Subscribers would then receive a live stream of I2RS interactions in trace log format and could flexibly choose to do a number of things with the log messages

## 2.2. Pub/Sub variants on Network Elements

This document is intended to cover requirements beyond I2RS. Looking at history, there are many examples of switching and routing protocols which have done explicit or implicit pub/sub in the past. In addition, new policy notification mechanisms which operate on switches and routers are being specified now. A small subset of current and past subscription mechanisms includes:

- Multicast topology establishment is accomplished before any content delivery is made to endpoints (IGMP, PIM, etc.)

- Secure Automation and Continuous Monitoring (SACM) allows subscription into devices which then may push spontaneous changes in their configured hardware and software[sacm-requirements]

- In MPLS VPNs [RFC6513] a Customer Edge router exchanges PIM control messages before PE Routing Adjacencies are passed. [RFC6513]

- After OSPF establishes its adjacencies, Link State Advertisement will then commence [RFC2328]

Worthy of note in the examples above is the wide variety of underlying transports. A generalized Pub/Sub mechanism therefore should be structured to support alternative transports. Based on current I2RS requirements, NETCONF should be the initially supported transport based on the need for connection-oriented/unicast communication. Eventual support for multicast and broadcast subscription update distribution will be needed as well.

## 2.3. Existing Generalized Pub/Sub Implementations

TIBCO, RSS, CORBA, and other technologies all show precursor Pub/Sub technologies. However there are new needs described in Section 4 below which these technologies do not serve. We need a new pub-sub technology.

There are at least two widely deployed generalized pub/sub implementations which come close to current needs: XMPP[XEP-0060] and DDS[OMG-DDS]. Both serve as proof-points that a highly scalable distributed datastore implementation connecting millions of edge devices is possible.

Because of these proof points, we can be comfortable that the underlying technologies can enable reusable generalized YANG object distribution. Analysis will need to fully dimension the speed and

scale of such object distribution for various subtree sizes and transport types.

## 3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. Although this document is not a protocol specification, the use of this language clarifies the instructions to protocol designers producing solutions that satisfy the requirements set out in this document.

A Subscriber makes requests for set(s) of YANG object data.

A Publisher is responsible for distributing subscribed YANG object data per the terms of a Subscription. In general, a Publisher is the owner of the YANG datastore that is subjected to the Subscription.

A Receiver is the target to which a Publisher pushes updates. In general, the Receiver and Subscriber will be the same entity. A Subscription Service provides Subscriptions to Subscribers of YANG data.

A Subscription Service interacts with the Publisher of the YANG data as needed to provide the data per the terms of the Subscription.

A Subscription Request for one or more YANG subtrees (including single leafs) is made by the Subscriber of a Publisher and is targeted to a Receiver. A Subscription may include constraints which dictate how often or under what conditions YANG information updates might be sent.

A Subscription is a contract between a Subscription Service and a Subscriber that stipulates the data to be pushed and the associated terms.

A datastore is defined in [RFC6241] and is not redefined here.

An Update provides object changes which have occurred within subscribed YANG subtree(s). An Update must include the current status of (data) node instances which according to any filtering are reportably different from the previously provided state. An Update may include a bundled set of ordered/sequential changes for a given object that have been made since the last update.

A Filter contains evaluation criteria which are evaluated against YANG object(s) within a Subscription. There are two types of Filters: Subtree Filters which identify selected objects/nodes

published under a target data node, and object element and attribute Filters where an object should only be published if it has properties meeting specified Filter criteria.

## 4. Requirements

Many of the requirements within this section have been adapted from XMPP[XEP-0060] and DDS[OMG-DDS] requirements specifications.

### 4.1. Assumptions for Subscriber Behavior

This document provides requirements for the Subscription Service. It does not define all the requirements for the Subscriber/Receiver. However in order to frame the desired behavior of the Subscription Service, it is important to specify key input constraints.

A Subscriber SHOULD avoid attempting to establish multiple Subscriptions pertaining to the same information, i.e. referring to the same datastore YANG subtrees.

A Subscriber MAY provide Subscription QoS criteria to the Subscription Service; if the Subscription Service is unable to meet those criteria, the Subscription SHOULD NOT be established.

When a Subscriber and Receiver are the same entity and the transport session is lost/terminated, the Subscriber MUST reestablish any subscriptions it previously created via signalling over the transport session. I.e., There is no requirement for the life span of such signaled Subscriptions extend beyond the life span of the transport session.

A Subscriber MUST be able to infer when a Subscription Service is no longer active and when no more updates are being sent.

A Subscriber MAY check with a Subscription Service to validate the existence and monitored subtrees of a Subscription.

A Subscriber MUST be able to periodically lease and extend the lease of a Subscription from a Subscription Service.

### 4.2. Subscription Service Requirements

#### 4.2.1. General

A Subscription Service MUST support the ability to create, renew, timeout, and terminate a Subscription.

A Subscription Service MUST be able to support and independently track multiple Subscription Requests by the same Subscriber.

A Subscription Service MUST be able to support an add/change/delete of subscriptions to multiple YANG subtrees as part of the same Subscription Request.

A Subscription Service MUST support Subscriptions against operational datastores, configuration datastores, or both.

A Subscription Service MUST be able support filtering so that subscribed updates under a target node might publish only operational data, only configuration data, or both.

A Subscription MAY include Filters as defined within a Subscription Request, therefore the Subscription Service MUST publish only data nodes that meet the Filter criteria within a Subscription.

A Subscription Service MUST support the ability to subscribe to periodic updates. The subscription period MUST be configurable as part of the subscription request.

A Subscription Service SHOULD support the ability to subscribe to updates "on-change", i.e., whenever values of subscribed data objects change.

For "on-change" updates, the Subscription Service MUST support a dampening period that needs to pass before the first or subsequent "on-change" updates are sent. The dampening period SHOULD be configurable as part of the subscription request.

A Subscription Service MUST allow Subscriptions to be monitored. Specifically, a Subscription Service MUST at a minimum maintain information about which Subscriptions are being serviced, the terms of those subscriptions (e.g., what data is being subscribed, associated Filters, update policy - on change, periodic), and the overall status of the Subscription - e.g., active or suspended.

A Subscription Service MUST support terminating of a Subscription when requested by the Subscriber.

A Subscription Service SHOULD support the ability to suspend and to resume a Subscription on request of a client.

A Subscription Service MAY at its discretion revoke or suspend an existing subscription. Reasons may include transitory resource limitation, credential expiry, failure to reconfirm a subscription, loss of connectivity with the Receiver, operator CLI, and/or others.

When this occurs, the Subscription Service MUST notify the Subscriber and update subscription status.

A Subscription Service MAY offer the ability to modify a subscription Filter. If such an ability is offered, the service MUST provide subscribers with an indication telling at what point the modified subscription goes into effect.

### 4.2.2. Negotiation

A Subscription Service MUST be able to negotiate the following terms of a Subscription:

- The policy: i.e. whether updates are on-change or periodic
- The interval, for periodic publication policy
- The dampening period, for on-change update policy (if supported)
- Any Filters associated with a subtree subscription

A Subscription Service SHOULD be able to negotiate QoS criteria for a Subscription. Examples of Subscription QoS criteria may include reliability of the Subscription Service, reaction time between a monitored YANG subtree/object change and a corresponding notification push, and the Subscription Service's ability to support certain levels of object liveliness.

In cases where a Subscription Request cannot be fulfilled, the Subscription Service MUST include in its decline a set of criteria that would have been acceptable when the Subscription Request was made. For example, if periodic updates were requested with too short update intervals for the specified data set, an alternative acceptable interval period might be returned from the Publisher. If on-change updates were requested with too-aggressive a dampening period, then an acceptable dampening period may be returned, or alternatively an indication that only periodic updates are supported for the requested object(s).

### 4.2.3. Update Distribution

For "on-change" updates, the Subscription Service MUST only send deltas to the object data for which a change occurred. [Otherwise the subscriber might not know what has actually undergone change.] The updates for each object MUST include an indication whether it was removed, added, or changed.

When a Subscription Service is not able to send updates per its subscription contract, the Subscription MUST notify subscribers and put the subscription into a state indicating the Subscription was suspended by the service. When able to resume service, subscribers need to be notified as well. If unable to resume service, the Subscription Service MAY terminate the subscription and notify Subscribers accordingly.

When a Subscription with "on-change" updates is suspended and then resumed, the first update SHOULD include updates of any changes that occurred while the Subscription was suspended, with the current value. The Subscription Service MUST provide a clear indication when this capability is not supported (because in this case a client application may have to synchronize state separately).

Multiple objects being pushed to a Subscriber, perhaps from different Subscriptions, SHOULD be bundled together into a single Update.

The sending of an Update MUST NOT be delayed beyond the Push Latency of any enclosed object changes.

The sending of an Update MUST NOT be delayed beyond the dampening period of any enclosed object changes.

The sending of an Update MUST NOT occur before the dampening period expires for any enclosed object changes.

A Subscription Service MAY, as an option, support a replay capability so that a set of updates generated during a previous time internal can be sent to a Receiver.

#### 4.2.4. Transport

A Subscription Service SHOULD support different transports.

A Subscription Service SHOULD support different encodings of payload.

It MUST be possible for Receivers to associate the update with a specific Subscription.

In the case of connection-oriented transport, when a transport connection drops, the associated Subscription SHOULD be terminated. It is up the Subscriber to request a new Subscription.

### 4.2.5. Security Requirements

As part of the Subscription establishment, there MUST be mutual authentication between the Subscriber and the Subscription Service.

When there are multiple Subscribers, it SHOULD be possible to provide cryptographic authentication in such a way that no Subscriber can pose as the original Subscription Service.

Versioning MUST be supported so that the capabilities and behaviors expected of specific technology implementations can be exposed.

A Subscription could be used to attempt to retrieve information that a client has not authorized access to. Therefore it is important that data pushed based on Subscriptions is authorized in the same way that regular data retrieval operations are authorized. Data being pushed to a client MUST be filtered accordingly, just like if the data were being retrieved on-demand. For Unicast transports, the NETCONF Authorization Control Model applies.

Additions or changes within a subscribed subtree structure MUST be validated against authorization methods before Subscription Updates including new subtree information are pushed.

A loss of authenticated access to subtree or node SHOULD be communicated to the Subscriber.

Subscription requests, including requests to create, terminate, suspend, and resume Subscriptions MUST be properly authorized.

When the Subscriber and Receiver are different, the Receiver MUST be able to terminate any Subscription to it where objects are being delivered over a Unicast transport.

A Subscription Service SHOULD decline a Subscription Request if it is likely to deplete its resources. It is preferable to decline a Subscription when originally requested, rather than having to terminate it prematurely later.

When the Subscriber and Receiver are different, and when the underlying transport connection passes credentials as part of transport establishment, then potentially pushed objects MUST be excluded from a push update if that object doesn't have read access visibility for that the Receiver.

#### 4.2.6. Subscription QoS

A Subscription Service SHOULD be able to negotiate the following Subscription QoS parameters with a Subscriber: Dampening, Reliability, Deadline, and Bundling.

A Subscription Service SHOULD be able to interpret Subscription QoS parameters, and only establish a Subscription if it is possible to meet the QoS needs of the provided QoS parameters.

##### 4.2.6.1. Liveliness

A Subscription Service MUST be able to respond to requests to verify the Liveliness of a subscription.

A Subscription Service MUST be able to report the currently monitored Nodes of a Subscription.

##### 4.2.6.2. Dampening

A Subscription Service MUST be able to negotiate the minimum time separation since the previous update before transmitting a subsequent update for Subscription. (Note: this is intended to confine the visibility of volatility into something digestible by the receiver.)

##### 4.2.6.3. Reliability

A Subscription Service MAY send Updates over Best Effort and Reliable transports.

##### 4.2.6.4. Coherence

For a particular Subscription, every update to a subscribed object MUST be sent to the Receiver in sequential order.

##### 4.2.6.5. Presentation

The Subscription Service MAY have the ability to bundle a set of discrete object notifications into a single publishable update for a Subscription. A bundle MAY include information on different Data Nodes and/or multiple updates about a single Data Node.

For any bundled updates, the Subscription Service MUST provide information for a Receiver to reconstruct the order and timing of updates.

#### 4.2.6.6. Deadline

The Subscription Service MUST be able to push updates at a regular cadence that corresponds with Subscriber specified start and end timestamps. (Note: the regular cadence can drive one, a discrete quantity, or an unbounded set of periodic updates.)

#### 4.2.6.7. Push Latency

The Subscription Service SHOULD be able to delay Updates on object push for a configurable period per Subscriber.

It MUST be possible for an administrative entity to determine the Push latency between object change in a monitored subtree and the Subscription Service Push of the update transmission.

#### 4.2.6.8. Relative Priority

The Subscription Service SHOULD include the relative priority of push updates so that dequeuing and discarding of case of limited bandwidth between Publisher and

### 4.2.7. Filtering

If no filtering criteria are provided, or if filtering criteria are met, updates for a subscribed object MUST be pushed, subject to the QoS limits established for the subscription.

It MUST be possible for the Subscription Service to receive Filter(s) from a Subscriber and apply them to corresponding object(s) within a Subscription.

It MUST be possible to attach one or more Subtree and/or object element and attribute Filters to a subscription. Mandatory Filter types include:

- For character-based object properties, Filter values which are exactly equal to a provided string, not equal to the string, or containing a string.

- For numeric based object properties, Filter values which are =, !=, <, <=, >, >= a provided number.

It SHOULD be possible for Filtering criteria to evaluate more than one property of a particular subscribed object as well as apply multiple Filters against a single object.

It SHOULD be possible to establish query match criteria on additional objects to be used in conjunction with Filtering criteria on a subscribed object. (For example: if A has changed and B=1, then Push A.) Query match capability may be done on objects within the datastore even if those objects are not included within the subscription. This of course assumes the subscriber has read access to those objects.

For on-change subscription updates, an object MUST pass a Filter through a Filter if it has changed since the previous update. This includes if the object has changed multiple times since the last update, and ithe value happens to be the exact same value as the last one sent.

### 4.2.8. Assurance and Monitoring

It MUST be possible to fetch the state of a single subscription from a Subscription Service.

It MUST be possible to fetch the state of all subscriptions of a particular Subscriber.

It MUST be possible to fetch a list and status of all Subscription Requests over a period of time. If there is a failure, some failure reasons might include:

- Improper security credentials provided to access the target node;
- Target node referenced does not exist;
- Subscription type requested is not available upon the target node;
- Out of resources, or resources not available;
- Incomplete negotiations with the Subscriber.

## 5. Security Considerations

There are no additional security considerations beyond the requirements listed in Section 4.2.5.

## 6. IANA Considerations

This document has no actions for IANA.

## 7. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Ambika Tripathy and Prabhakara Yellai as well as the helpfulness of related end-to-end system context info from Nancy Cam Winget, Ken Beck, and David McGrew.

## 8. References

### 8.1. Normative References

[i2rs-arch]
Atlas, A., "An Architecture for the Interface to the Routing System", February 2016, <https://datatracker.ietf.org/doc/draft-ietf-i2rs-architecture/>.

[i2rs-traceability]
Clarke, J., Salgueiro, G., and C. Pignataro, "Interface to the Routing System (I2RS) Traceability: Framework and Information Model", February 2016, <https://datatracker.ietf.org/doc/draft-ietf-i2rs-traceability/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, DOI 10.17487/RFC2328, April 1998, <http://www.rfc-editor.org/info/rfc2328>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC6513] Rosen, E., Ed. and R. Aggarwal, Ed., "Multicast in MPLS/ BGP IP VPNs", RFC 6513, DOI 10.17487/RFC6513, February 2012, <http://www.rfc-editor.org/info/rfc6513>.

### 8.2. Informative References

[i2rs-usecase]
Hares, S. and M. Chen, "Summary of I2RS Use Case Requirements", March 2016, <https://datatracker.ietf.org/doc/draft-ietf-i2rs-usecase-reqs-summary/>.

[OMG-DDS] "Data Distribution Service for Real-time Systems, version 1.2", January 2007, <http://www.omg.org/spec/DDS/1.2/>.

[sacm-requirements]
Cam Winget, N., "Secure Automation and Continuous Monitoring (SACM) Requirements", March 2016, <https://tools.ietf.org/html/draft-ietf-sacm-requirements-09>.

[XEP-0060]
Millard, P., "XEP-0060: Publish-Subscribe", July 2010, <XEP-0060: Publish-Subscribe>.

## Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alexander Clemm
Cisco Systems

Email: alex@cisco.com

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com