NETCONF
Internet-Draft
Intended status: Informational
Expires: December 17, 2016

E. Voit
A. Clemm
A. Tripathy
E. Nilsen-Nygaard
A. Gonzalez Prieto
Cisco Systems
June 15, 2016

Restconf and HTTP Transport for Event Notifications
draft-voit-netconf-restconf-notif-00

Abstract

This document defines Event Notification YANG Subscription and Push mechanisms for Restconf, HTTP, and HTTP2 transports.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Mechanisms to support Event subscription and push are defined in [rfc5277bis]. Enhancements to [rfc5277bis] which enable YANG Datastore subscription and push are defined in [yang-push]. This document provides a transport specification for these Restconf and HTTP. This has been driven by Requirements for subscriptions to YANG datastores are defined in [pub-sub-reqs].

Beyond based transport bindings, there are benefits which can be realized when transporting updates directly HTTP/2[RFC7540] which cn be realized via an implementation of this transport specification including:

o Subscription multiplexing over independent HTTP/2 streams

o Stream prioritization and stream dependencies

o Flow control on independent streams

2. 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 RFC 2119 [RFC2119].

Configured Subscription: a Subscription installed via a configuration interface which persissts across reboots.

Data Node: An instance of management information in a datastore.

Data Node Update: A data item containing the current value/property of a Data Node at the time the Data Node Update was created.

Dynamic Subscription: a Subscription negotiated between Subscriber and Publisher via create, establish, modify, and delete RPC control plane signaling messages.

Event: an occurrence of something that may be of interest. (e.g., a configuration change, a fault, a change in status, crossing a threshold, status of a flow, or an external input to the system.)

Event Notification: a set of information intended for a Receiver indicating that one or more Event(s) have occurred. Details of the Event(s) may be included within.

Event Stream: a continuous, ordered set of Events grouped under an explicit criteria.

Notification: the communication of an occurrence, perhaps triggered by the occurrence of an Event.

Publisher: an entity responsible for streaming Event Notifications per the terms of a Subscription.

Receiver: a target to which a Publisher pushes Event Notifications. For Dynamic Subscriptions, the Receiver and Subscriber will often be the same entity.

Subscriber: an entity able to request and negotiate a contract for the receipt of Event Notifications from a Publisher

Subscription: a contract between a Subscriber and a Publisher stipulating which information the Receiver wishes to have pushed from the Publisher without the need for further solicitation.

3. Solution

Event subscription is defined in [rfc5277bis], YANG Datastore subscription is defined in [yang-push]. This section specifies transport mechanisms applicable to both.

3.1. Mechanisms for Subscription Establishment and Maintenance

There are three models for Subscription establishment and maintenance:

1. Dynamic Subscription: Here the Subscriber and Receiver are the same. A Subscription ends with a subscription-terminated notification, or by a loss of transport connectivity.

2. Configured Subscription: Receiver(s) are specified on Publisher in startup and running config. Subscription is not terminated except via an operations interface. (Subscriptions may be Suspended, with no Event Notifications sent however.)

3. Proxy Subscription: Subscriber and Receiver are different. Subscription ends when a Subscription End-time is reached, or the Publisher process is restarted.

The first two are described in this section. The third is described in Appendix A. This third option can be moved into the body of this specification should the IETF community desire. In theory, all three models may be intermixed in a single deployment.

```
                          .---------------.
                          |   Publisher   |
                          '---------------'
                            ^   ^   |   ^
                            |   |   |   |
          .-----Restconf----'   |   |   '-----Restconf----.
          |               .-----'   '-HTTP-.              |
          V               |                V              |
   .-------------.  .------------.  .----------.  .------------.
   | Subscriber+ |  | Operations |  | Receiver |  | Subscriber |
   | Receiver    |  |  /Config   |  '----------'  '------------'
   '-------------'  '------------'    ^     ^               ^
          ^         (out of scope)    :     :               :
          :             ^             :     :....Model 3....:
        Model 1         :...Model 2...:      (out of scope)
```

3.1.1. Dynamic YANG Subscription over RESTCONF

Dynamic Subscriptions are configured and manage Subscriptions via signaling. This signaling is transported over [restconf]. Once established, streaming Event Notifications are then delivered via Restconf SSE.

3.1.2. Configured Subscription over HTTP

With a Configured Subscription, all information needed to establish a secure relationship with that Receiver is configured on the Publisher. With this information, the Publisher will establish a secure transport connection with the Receiver and then begin pushing the Event Notifications to the Receiver. Since Restconf might not exist on the Receiver, it is not desirable to require that such Event Notifications be pushed via Restconf. Nor is there value which Restconf provides on top of HTTP. Therefore in place of Restconf, a TLS secured HTTP Client connection must be established with an HTTP Server located on the Receiver. Event Notifications will then be sent via HTTP Post messages to the Receiver.

Post messages will be addressed to HTTP augmentation code on the Receiver capable of accepting and responding to Event Notifications. At least the initial Post message must include the URI for the subscribed resource. This URI can be retained for operational tracking and debugging use by the Receiver.

After successful receipt of an initial Event Notification for a particular Subscription, the Reciever should reply back with an HTTP status code of 201 (Created). Further successful receipts should result in the return of code of 202 (Accepted). At any point, receipt of any status codes from 300-510 with the exception of 408 (Request Timeout) should result in the movement of the Subscription to the suspended state. A sequential series of multiple 408 exceptions should also drive the Subscription to a suspended state.

Security on an HTTP client/Publisher can be strengthened by only accepting Response code feedback for recently initiated HTTP POSTs.

Figure 2 depicts this message flow.

```
      +-----------+               +----------+
      | Publisher |               | Receiver |
      +-----------+               +----------+
           |<--------------TLS------------>|
           |                               |
           |HTTP POST (Sub ID, URI, data1) |
           |------------------------------>|
           |             HTTP 201 (Created)|
           |<------------------------------|
           |HTTP POST (Sub ID, data2)      |
           |------------------------------>|
           |     HTTP 200 or 202 (Accepted)|
           |<------------------------------|
           |             data3             |
           |<----------------------------->|
```

If HTTP/2 transport is available to a Receiver, the Publisher should also:

- o point individual Event Notifications to a unique HTTP/2 stream for that Subscription,
- o take any subscription-priority and provision it into the HTTP/2 stream priority, and
- o take any subscription-dependency and provision it into the HTTP/2 stream dependency.

3.2. Subscription Multiplexing

When pushed directly over HTTP/2, it is expected that the Event Notifications from a single Subscription will be allocated a separate HTTP/2 stream. This will enable multiplexing, and address issues of Head-of-line blocking with different priority Subscriptions.

When pushed via Restconf over HTTP/2, different Subscriptions will not be mapped to independent HTTP/2 streams. When Restconf specifies this mapping, support may be appended on top of this specification.

With or without independent queueing of multiplexed subscriptions, it is possible that updates might be delivered in a different sequence than generated. Reasons for this might include (but are not limited to):

- o replay of pushed updates
- o temporary loss of transport connectivity, with update buffering and different dequeuing priorities per Subscription

- population, marshalling and bundling of independent Subscription Updates, and

- parallel HTTP1.1 sessions

Therefore each Event Notification will include a microsecond level timestamp to ensure that a Receiver understands the time when a that update was generated. Use of this timestamp can give an indication of the state of objects at a Publisher when state-entangled information is received across different subscriptions. The use of the latest Event Notification timestamp for a particular object update can introduce errors. So when state-entangled updates have inconsistent object values and temporally close timestamps, a Receiver might consider performing a 'get' to validate the current state of a Publisher.

3.3. Push Data Stream and Transport Mapping

Transported updates will contain data for one or more Event Notifications. Each transported Event Notification will contain several parameters:

- A Subscription ID correlator

- Event Notification(s) . (Note 1: These must be filtered per access control rules to contain only data that the Subscriber is authorized to see. Note 2: these Event Notifications might be Data Node Update(s).)

- A timestamp indication when the Event Notification was generated on the Publisher.

3.3.1. Subscription and Updates via Restconf

Subscribers can dynamically learn whether a RESTCONF server supports various types of Event or Yang datastore subscription. This is done by issuing an HTTP request OPTIONS, HEAD, or GET on the stream. Some examples building upon the existing RESTCONF mechanisms are below:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/
         streams/stream=yang-push HTTP/1.1
Host: example.com
Accept: application/yang.data+xml
```

If the server supports it, it may respond

```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
<stream xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
            <name>yang-push</name>
            <description>Yang push stream</description>
            <access>
               <encoding>xml</encoding>
               <location>https://example.com/streams/yang-push-xml
               </location>
            </access>
            <access>
               <encoding>json</encoding>
               <location>https://example.com/streams/yang-push-json
               </location>
            </access>
         </stream>
```

If the server does not support any form of subscription, it may respond

```
HTTP/1.1 404 Not Found
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
```

Subscribers can determine the URL to receive updates by sending an HTTP GET request for the "location" leaf with the stream list entry.The stream to use for may be selected from the Event Stream list provided in the capabilities exchange. Note that different encodings are supporting using different Event Stream locations. For example, the Subscriber might send the following request:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/
         streams/stream=yang-push/access=xml/location HTTP/1.1
Host: example.com
Accept: application/yang.data+xml
```

The publisher might send the following response:

```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
   <location
        xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
        https://example.com/streams/yang-push-xml
   </location>
```

To subscribe and start receiving updates, the subscriber can then send an HTTP GET request for the URL returned by the Publisher in the request above. The accept header must be "text/event-stream". The

Publisher uses the Server Sent Events[W3C-20121211] transport strategy to push filtered Event Notifications from the Event stream,.

The publisher MUST support as query parameters for a GET method on this resource all the parameters of a subscription. The only exception is the encoding, which is embedded in the URI. An example of this is:

```
// subtree filter = /foo
// periodic updates, every 5 seconds
GET /mystreams/yang-push?subscription-id=my-sub&period=5&
           xpath-filter=%2Fex:foo[starts-with("bar"."some"]
```

Should the publisher not support the requested subscription, it may reply:

```
HTTP/1.1 501 Not Implemented
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+xml
    <errors xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
       <error>
           <error-type>application</error-type>
           <error-tag>operation-not-supported</error-tag>
           <error-severity>error</error-severity>
           <error-message>Xpath filters not supported</error-message>
           <error-info>
               <supported-subscription xmlns="urn:ietf:params:xml:ns:
                   netconf:datastore-push:1.0">
                   <subtree-filter/>
               </supported-subscription>
           </error-info>
       </error>
     </errors>
```

with an equivalent JSON encoding representation of:

```
HTTP/1.1 501 Not Implemented
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+json
      {
        "ietf-restconf:errors": {
          "error": {
            "error-type": "protocol",
            "error-tag": "operation-not-supported",
            "error-message": "Xpath filters not supported."
            "error-info": {
               "datastore-push:supported-subscription": {
                     "subtree-filter": [null]
                 }
            }
          }
        }
      }
```

The following is an example of a pushed Event Notification data for the subscription above. It contains a subtree with root foo that contains a leaf called bar:

XML encoding representation:

```
  <?xml version="1.0" encoding="UTF-8"?>
  <notification xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
     <subscription-id xmlns="urn:ietf:params:xml:ns:restconf:
         datastore-push:1.0">
           my-sub
     </subscription-id>
     <eventTime>2015-03-09T19:14:56Z</eventTime>
     <datastore-contents xmlns="urn:ietf:params:xml:ns:restconf:
        datastore-push:1.0">
        <foo xmlns="http://example.com/yang-push/1.0">
          <bar>some_string</bar>
        </foo>
     </datastore-contents>
  </notification>
```

Or with the equivalent YANG over JSON encoding representation as defined in[yang-json] :

```
{
  "ietf-restconf:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56Z",
    "datastore-push:datastore-contents": {
      "example-mod:foo": { "bar": "some_string" }
    }
  }
}
```

To modify a Subscription, the subscriber issues another GET request on the provided URI using the same subscription-id as in the original request. For example, to modify the update period to 10 seconds, the subscriber may send:

```
GET /mystreams/yang-push?subscription-id=my-sub&period=10&
       subtree-filter=%2Ffoo'
```

To delete a Subscription, the Subscriber issues a DELETE request on the provided URI using the same subscription-id as in the original request

```
DELETE /mystreams/yang-push?subscription-id=my-sub
```

3.3.2. Subscription and Updates directly via HTTP

For any version of HTTP, the basic encoding will look as below. It consists of a JSON representation wrapped in an HTTP header.

```
POST (IP+Port) HTTP/1.1
From: (Identifier for Network Element)
User-Agent: (CiscoYANGPubSub/1.0)
Content-Type: multipart/form-data
Content-Length: (determined runtime)
{
  "ietf-yangpush:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56Z",
    "datastore-push:datastore-contents": {
      "foo": { "bar": "some_string" }
    }
  }
}
```

4. Security Considerations

Subscriptions could be used to intentionally or accidentally overload resources of a Publisher. For this reason, it is important that the Publisher has the ability to prioritize the establishment and push of Event Notifications where there might be resource exhaust potential. In addition, a server needs to be able to suspend existing Subscriptions when needed. When this occurs, the subscription status must be updated accordingly and the Receivers notified.

A Subscription could be used to attempt retrieve information for which a Receiver has no authorized access. Therefore it is important that data pushed via a Subscription is authorized equivalently with regular data retrieval operations. Data being pushed to a Receiver needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model [RFC6536] applies even though the transport is not NETCONF.

One or more Publishers could be used to overwhelm a Receiver which doesn't even support Subscriptions. Therefore Event Notifications for Configured Subscriptions MUST only be transmittable over Encrypted transports. Clients which do not want pushed Event Notifications need only terminate or refuse any transport sessions from the Publisher.

One or more Publishers could overwhelm a Receiver which is unable to control or handle the volume of Event Notifications received. In deployments where this might be a concern, transports supporting per-subscription Flow Control and Prioritization (such as HTTP/2) should be selected.

Another benefit is that a well-behaved Publisher implementation is that it is difficult to a Publisher to perform a DoS attack on a Receiver. DoS attack protection comes from:

- o the requirement for trust of a TLS session before publication,
- o the need for an HTTP transport augmentation on the Receiver, and
- o that the Publication process is suspended when the Receiver doesn't respond.

5. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Andy Bierman, Sharon Chisholm, Yan Gang, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Hector Trevino, Kent Watsen, and Guangying Zheng.

6. References

6.1. Normative References

[restconf]
Bierman, Andy., Bjorklund, Martin., and Kent. Watsen, "RESTCONF Protocol", March 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf/>.

[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>.

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <http://www.rfc-editor.org/info/rfc6520>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

6.2. Informative References

[call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", December 2015, <https://tools.ietf.org/html/draft-ietf-netconf-call-home-17>.

[pub-sub-reqs]
Voit, Eric., Clemm, Alexander., and Alberto. Gonzalez Prieto, "Subscribing to datastore push updates", February 2016, <https://datatracker.ietf.org/doc/draft-ietf-i2rs-pub-sub-requirements/>.

[rfc5277bis]
Gonzalez Prieto, Alberto., Clemm, Alexander., Voit, Eric., Prasad Tripathy, Ambika., and Einar. Nilsen-Nygaard, "NETCONF Event Notifications", March 2016, <https://datatracker.ietf.org/doc/draft-gonzalez-netconf-5277bis/>.

[W3C-20121211]
"Server-Sent Events, World Wide Web Consortium CR CR-eventsource-20121211", December 2012, <http://www.w3.org/TR/2012/CR-eventsource-20121211>.

[yang-json]
Lhotka, Ladislav., "JSON Encoding of Data Modeled with YANG", March 2016, <https://datatracker.ietf.org/doc/draft-ietf-netmod-yang-json/>.

[yang-push]
Clemm, Alexander., Gonzalez Prieto, Alberto., Voit, Eric., Prasad Tripathy, Ambika., and Einar. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", February 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

Appendix A. Proxy YANG Subscription when the Subscriber and Receiver are different

The properties of Dynamic and Configured Subscriptions can be combined to enable deployment models where the Subscriber and Receiver are different. Such separation can be useful with some combination of:

o An operator doesn't want the subscription to be dependent on the maintenance of transport level keep-alives. (Transport independence provides different scalability characteristics.)

o There is not a transport session binding, and a transient
   Subscription needs to survive in an environment where there is
   unreliable connectivity with the Receiver and/or Subscriber.

o An operator wants the Publisher to include highly restrictive
   capacity management and Subscription security mechanisms outside
   of domain of existing operational or programmatic interfaces.

To build a Proxy Subscription, first the necessary information must
be signaled as part of the <establish-subscription>. Using this set
of Subscriber provided information; the same process described within
section 3 will be followed. There is one exception. When an HTTP
status code is 201 is received by the Publisher, it will inform the
Subscriber of Subscription establishment success via its Restconf
connection.

After a successful establishment, if the Subscriber wishes to track
the state of Receiver subscriptions, it may choose to place a
separate on-change Subscription into the "Subscriptions" subtree of
the YANG Datastore on the Publisher.

Putting it all together, the message flow is:

```
        +------------+          +-----------+            +----------+
        | Subscriber |          | Publisher |            | Receiver |
        +------------+          +-----------+            +----------+
           | Restconf PUT:            |                            |
           | <establish-subscription>|                            |
           |------------------------>|                            |
           |                         |                            |
           |                         |<-----------TLS------------>|
           |                         |                            |
           |                         |HTTP POST (Sub ID, data1,   |
           |                         |(stream ID, URI?))          |
           |                         |--------------------------->|
           |                         |          HTTP 201 (Created)|
           |                         |<---------------------------|
           |        Success: HTTP 204|                            |
           |<------------------------|                            |
           |                         |HTTP POST (Sub ID, data2)   |
           |                         |--------------------------->|
           |                         |  HTTP 200 or 202 (Accepted)|
           |                         |<---------------------------|
           |                         |            data3           |
           |                         |<-------------------------->|
           |                         |                            |
```

Appendix B. End-to-End Deployment Guidance

Several technologies are expected to be seen within a deployment to achieve security and ease-of-use requirements. These are not necessary for an implementation of this specification, but will be useful to consider when considering the operational context.

B.1. Call Home

Pub/Sub implementations should have the ability to transparently incorporate lower layer technologies such as Call Home so that secure TLS connections are always originated from the Publisher. There is a Restconf Call home function in [call-home]. For security reasons, this should be implemented when applicable.

B.2. TLS Heartbeat

Unlike NETCONF, HTTP sessions might not quickly allow a Subscriber to recognize when the communication path has been lost from the Publisher. To recognize this, it is possible for a Receiver (usually the subscriber) to establish a TLS heartbeat [RFC6520]. In the case where a TLS heartbeat is included, it should be sent just from Receiver to Publisher. Loss of the heartbeat should result in the Subscription being terminated with the Subscriber (even when the Subscriber and Receiver are different). The Subscriber can then attempt to re-establish the subscription if desired. If the Subscription remains active on the Publisher, future receipt of objects associated with that (or any other unknown) subscription ID should result in a <delete-subscription> being returned to the Publisher from the Receiver.

Appendix C. Issues being worked and resolved

(To be removed by RFC editor prior to publication)

C.1. Unresolved Issues

RT1 - Integration specifics for Restconf capability discovery on different types of Streams

RT2 - In what way to we position "Event notifications" model in this document vs. current solution in Restconf.

RT3 - Do we include 3rd party signaled subscriptions within models that need to be supported generically, or for a particular type of transport.

RT6 - We need to define encodings of rfc5277bis notifications for both Restconf and HTTP.

RT7 - HTTP native option doesn't currently use SSE. But we should evaluate moving to that as possible. It will make development integration easier and more consistent.

C.2. Agreement in principal

RT4 - Need to add into document examples of 5277bis Event streams. Document only includes yang-push examples at this point.

C.3. Resolved Issues

RT5 - Doesn't make sense to use Restconf for Configured subscriptions. HTTP will be used.

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alexander Clemm
Cisco Systems

Email: alex@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com