NETCONF
Internet-Draft
Intended status: Informational
Expires: September 18, 2016

E. Voit
A. Clemm
A. Tripathy
E. Nilsen-Nygaard
A. Gonzalez Prieto
Cisco Systems
March 17, 2016

# Restconf, HTTP, and HTTP2 Transport for YANG Push
# draft-voit-netconf-restconf-yang-push-02

## Abstract

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

## Status of This Memo

## Copyright Notice

Table of Contents

## 1. Introduction

Requirements for subscriptions to YANG datastores are defined in [pub-sub-reqs]. Mechanisms to support YANG subscriptions and datastore object push over a NETCONF are defined in [yang-push]. Restconf support is also needed by the market. This document provides such a specification for Restconf by reusing the YANG data model, and expanding the transport requirements of [yang-push].

These extensions are not limited to just Restconf. There are benefits which can be realized when transporting push updates directly over HTTP such as simplified support for static subscriptions. Additionally if HTTP/2 [RFC7540] transport is used, HTTP/2 capabilities which can be applied include:

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].

Datastore: a conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

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

Publisher: an entity 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.

Receiver: a target to which Publisher pushes updates. In many deployments, the Receiver and Subscriber will be the same entity.

Static Subscription: A Subscription installed via a configuration interface.

Subscriber: an entity able to request and negotiate a contract for push updates 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.

Subscription Update: a set of data nodes and object values pushed together as a unit and intended to meet the obligations of a single subscription at a snapshot in time.

## 3. Solution

This document specifies transport mechanisms that allow subscribed information updates to be pushed from a YANG datastore.

### 3.1. Subscription Model

Subscriptions use the base data model and subscription state machine from [yang-push].

### 3.2. Mechanisms for Subscription Establishment and Maintenance

On a Publisher, it must be possible to instantiate a Subscription via dynamic Subscriber signaling, as well as via Static configuration.

Dynamic Subscriptions are signaled Subscriptions aimed at the running datastore and are unable to impact the startup configuration. They

should always terminate when there is loss of transport session connectivity between the Publisher and Receiver.

Static Subscriptions are applied via an operations interface to the startup and running configurations. Loss or non-availability of transport session connectivity will place the Subscription into the suspended state. Logic beyond the scope of this specification will dictate when any particular Subscription should be reactivated. There are three models for Subscription establishment and maintenance:

1. Dynamic Subscription: Subscriber and Receiver are the same

2. Static Subscription

3. Dynamic Subscription: Subscriber and Receiver are different

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)
```

### 3.2.1. Dynamic YANG Subscription: Subscriber and Receiver are the same

With all Dynamic Subscriptions, as with [yang-push] it must be possible to configure and manage Subscriptions via signaling. This signaling is transported over [restconf]. Once established, streaming Subscription Updates are then delivered via Restconf SSE.

### 3.2.2. Static Subscription

With a Static Subscription, all information needed to establish a secure object push relationship with that Receiver must be configured via a configuration interface on the Publisher. This information includes all the signaled information identified in section 3.2.1. This information also include the Receiver address, egress interface instructions, and security credentials required to establish TLS between the Publisher and Receiver. Mechanisms for locally configuring these parameters are outside the scope of this document.

With this information, the Publisher will establish a secure transport connection with the Receiver and then begin pushing the streaming updates to the Receiver. Since Restconf might not exist on the Receiver, it is not desirable to require that updates be pushed via Restconf. In place of Restconf, a TLS secured HTTP Client connection must be established with an HTTP Server located on the Receiver. Subscription Updates will then be sent via HTTP Post messages to the Receiver.

Post messages will be addressed to HTTP augmentation code on the Receiver capable accepting and responding to Subscription Updates. At least the initial Post message must include the URI for the subscribed resource. This URI can be retained for future use by the Receiver.

After successful receipt of an initial Subscription Update for a particular Subscription, this augmentation 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 3 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 Subscription Updates 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.

## 3.3. Subscription Multiplexing

When pushed directly over HTTP/2, it is expected that each Subscription Update will be allocated a separate Stream.  The 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, it should be integrated into this specification.

Even without HTTP/2 multiplexing, 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  different durations needed to create various Subscription Updates,
- o  marshalling and bundling of multiple Subscription Updates for transport, and

- parallel HTTP1.1 sessions

Therefore each Subscription Update 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 Subscription Update 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 objects.

## 3.4. Push Data Stream and Transport Mapping

Transported updates will contain data for one or more Subscription Updates. Each transported Subscription Update notification contains several parameters:

- A global subscription ID correlator, referencing the name of the Subscription on whose behalf the notification is sent.

- Data nodes containing a representation of the datastore subtree containing the updates. The set of data nodes must be filtered per access control rules to contain only data that the subscriber is authorized to see.

- An event time which contains the time stamp at publisher when the event is generated.

### 3.4.1. Pushing Subscription Updates via Restconf

Subscribers can dynamically learn whether a RESTCONF server supports yang-push. This is done by issuing an HTTP request OPTIONS, HEAD, or GET on the stream push-update. E.g.:

```
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 yang push, 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 yang push is the push-update stream.  The location returned by the publisher can be used for the actual notification subscription.  Note that different encodings are supporting using different locations.  For example, he 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 handles the connection as an event stream, using the Server Sent Events[W3C-20121211] transport strategy.

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 push Subscription Update 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
```

### 3.4.2. Pushing Subscription Updates directly via HTTP

For any version of HTTP, the basic encoding will look as below is the above JSON representation wrapped in an HTTP header. Mechanism will be

```
   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 updates 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 clients are notified.

A Subscription could be used to retrieve data in subtrees that a client has not authorized access to. Therefore it is important that data pushed via a Subscription is authorized equivalently with regular data retrieval operations. Data being pushed to a client 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 Updates MUST only be transmittable over Encrypted transports. Clients which do not want pushed data 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 Updates 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. References

### 5.1. Normative References

[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>.

### 5.2. 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/>.

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

[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. Dynamic YANG Subscription when the Subscriber and Receiver are different

The methods of Sections 3.2.1 and 3.2.2 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 wants any Subscriptions immediately deleted should TLS connectivity be lost. (I.e., Subscriptions don't default into a 'Suspended' state on the Publisher.)

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

o Restconf is not desired on the Receiver.

o The Publisher doesn't want to maintain Restconf subscriptions with many Receivers.

To do this, first the necessary information must be signaled as part of the <create-subscription>. This includes all the information described in section 3.3.2, with the exception of the security credentials. (It is assumed that any security credentials required

for establishing any transport connections are pre-provisioned on all devices.)

Using this set of Subscriber provided information, the same process described within section 3.3.2 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 successful establishment, if the Subscriber wishes to maintain the state of Receiver subscriptions, it can simply 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:         |                             |
          | <create-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.

### B.3. Putting it together

If Subscriber and receiver are same entity then subscriber can direct send create_subscription message to publisher. Once the subscription moved to accepted state, the receiver can use Server Sent Events [W3C-20121211] transport strategy to subscriber event notifications for the data as defined in [restconf].

## 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