NETCONF
Internet-Draft
Intended status: Standards Track
Expires: February 5, 2018

E. Voit
A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
A. Clemm
Huawei
A. Gonzalez Prieto
VMWare
A. Bierman
YumaWorks

August 4, 2017

Restconf and HTTP Transport for Event Notifications
draft-ietf-netconf-restconf-notif-03

Abstract

This document defines Restconf, HTTP2, and HTTP1.1 bindings for the transport of Subscription requests and corresponding push updates. Being subscribed may be either publisher defined event streams or nodes/subtrees of YANG Datastores.

Table of Contents

1. Introduction

Mechanisms to support Event subscription and push are defined in [sn]. Enhancements to [sn] which enable YANG Datastore subscription and push are defined in [yang-push]. This document provides a transport specification for these protocols over Restconf and HTTP. Driving these requirements is [RFC7923].

The streaming of notifications encapsulating the resulting information push can be done with either HTTP1.1 and HTTP2. When using HTTP2 [RFC7540] benefits which can be realized include:

- Elimination of head-of-line blocking

- Weighting and proportional dequeuing of Events from different subscriptions

- Explicit precedence in subscriptions so that events from one subscription must be sent before another dequeues

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].

The following terms use the defintions from [sn]: configured subscription, dynamic subscription, event notification, publisher, receiver, subscriber, and subscription.

3. Solution

Subscribing to event streams is defined in [sn], YANG Datastore subscription is defined in [yang-push]. This section specifies transport mechanisms applicable to both.

3.1. Dynamic YANG Subscription with RESTCONF control

Dynamic Subscriptions for both [sn] and its [yang-push] augmentations are configured and managed via signaling messages transported over [RFC8040]. These interactions will be accomplished via a Restconf POST into RPCs located on the Publisher. HTTP responses codes will indicate the results of the interaction with the Publisher. An HTTP status code of 200 is the proper response to a successful <establish-subscription> RPC call. The successful <establish-subscription> will result in a HTTP message with returned subscription URI on a logically separate mechanism than was used for the original Restconf POST. This mechanism is via a parallel TCP connection in the case of HTTP 1.x, or in the case of HTTP2 via a separate HTTP stream within the HTTP connection. When a being returned by the Publisher, failure will be indicated by error codes transported in payload.

Once established, the resulting stream of notification messages are then delivered via SSE for HTTP1.1 and via HTTP Data for HTTP2.

3.1.1. Call Flow for HTTP2

Requests to [sn] or [yang-push] augmented RPCs are sent on one or more HTTP2 streams indicated by (a) in Figure 2. Notification messages related to a single subscription are pushed on a unique logical channel (b). In the case below, a newly established subscription has its associated messages pushed over HTTP2 stream (7).

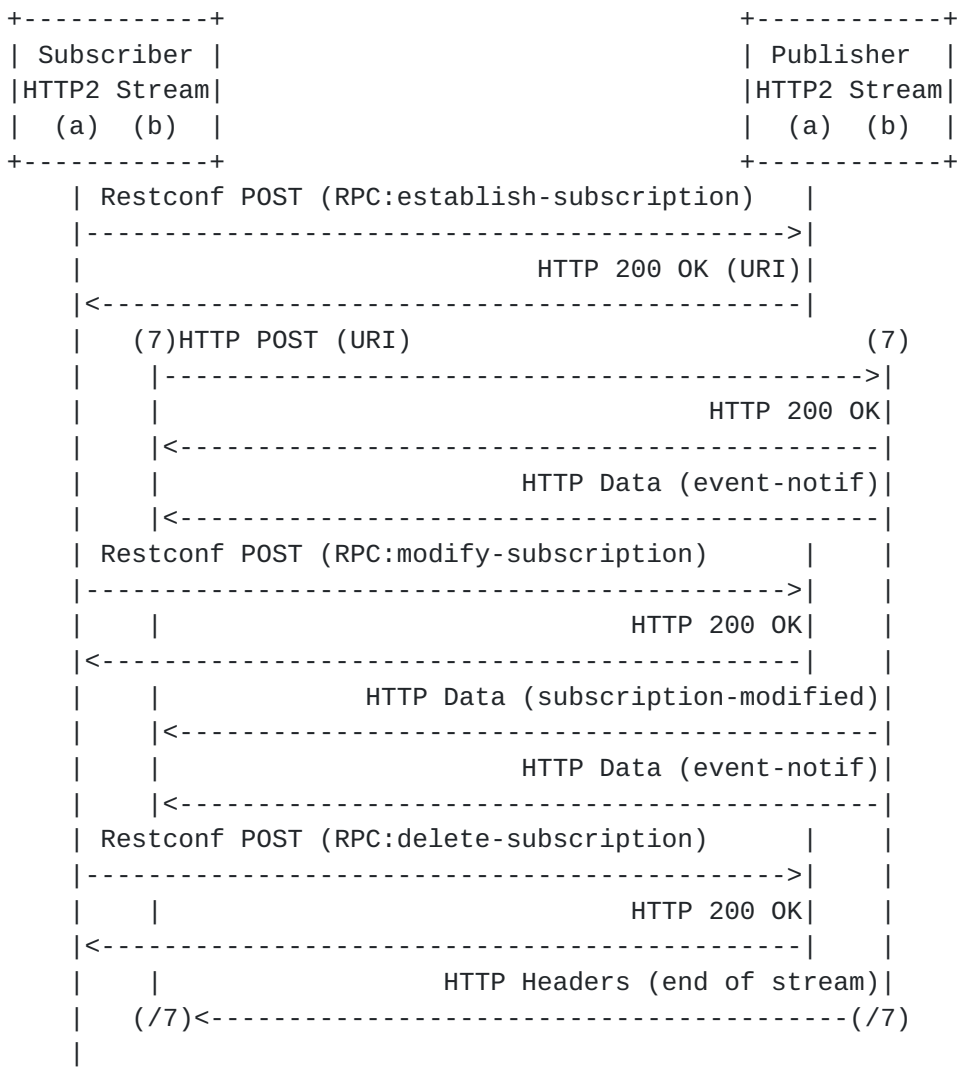

Figure 1: Dynamic with HTTP2

3.1.2. Call flow for HTTP1.1

Requests to [yang-push] RPCs are sent on the TCP connection indicated by (a). Notification messages are pushed on a separate connection (b). This connection (b) will be used for all notification messages across all subscriptions.

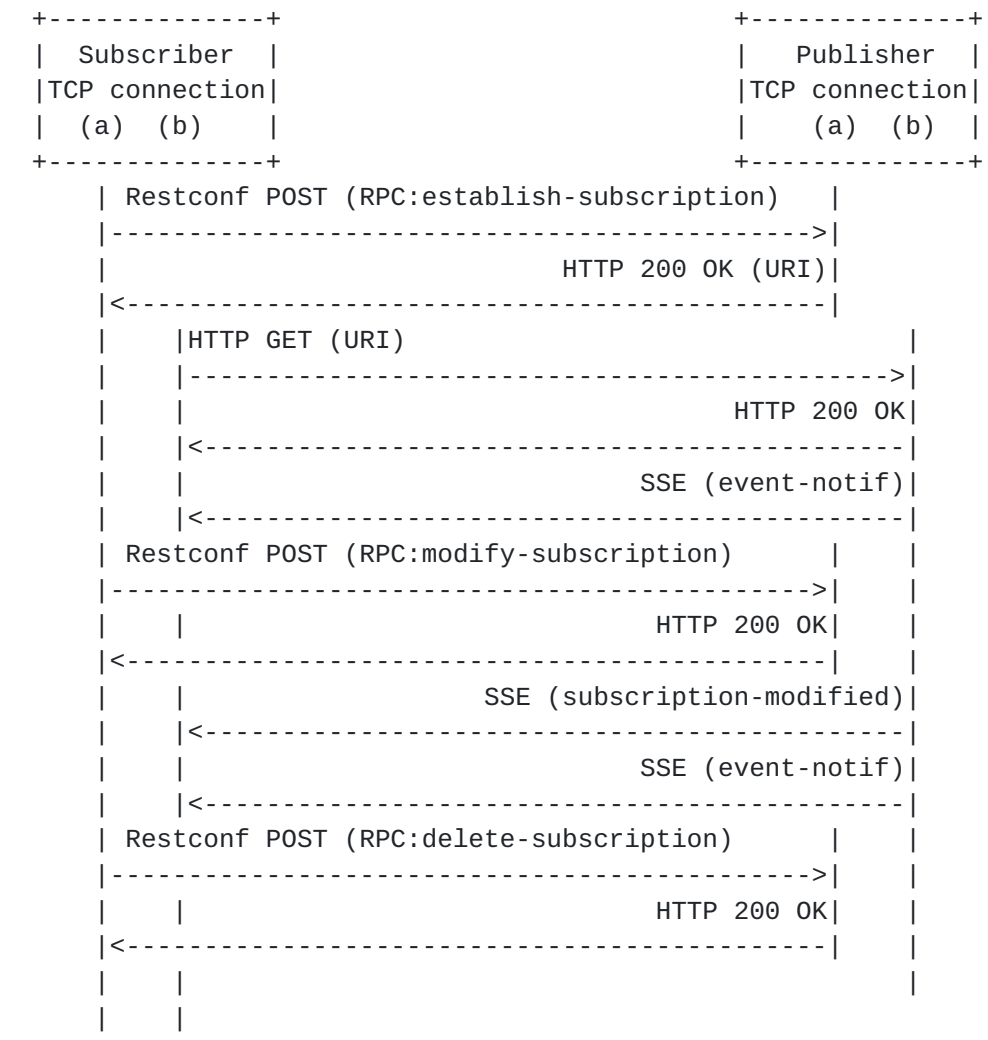

Figure 2: Dynamic with HTTP1.1

3.1.3. Configured Subscription over HTTP2

With a Configured Subscription, all information needed to establish a secure relationship with that Receiver is available on the Publisher. With this information, the Publisher will establish a secure transport connection with the Receiver and then begin pushing notification messages to the Receiver. Since Restconf might not exist on the Receiver, it is not desirable to require that subscribed content be pushed with any dependency on Restconf. Nor is there value which Restconf provides on top of HTTP. Therefore in place of Restconf, a TLS secured HTTP2 Client connection must be established with an HTTP2 Server located on the Receiver. Notification messages will then be sent as part of an extended HTTP POST to the Receiver.

POST messages will be addressed to HTTP augmentation code on the Receiver capable of accepting and responding to state change

notifications and subscribed content notification messages. The first POST message must be a subscription-started notification. Notifications which include any subscribed content must not be sent until the receipt of an HTTP 200 OK for this initial notification. The 200 OK will indicate that the Receiver is ready for the delivery of subscribed content. At this point a Subscription must be allocated its own HTTP2 stream. Figure 4 depicts this message flow.

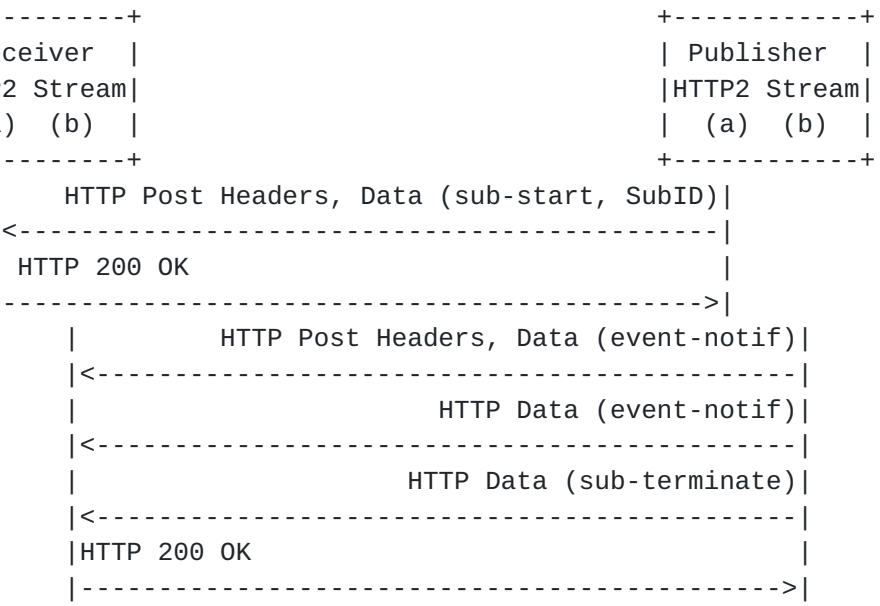

Figure 3: Configured over HTTP2

As the HTTP2 transport is available to the Receiver, the Publisher should:

- take any subscription-priority and copy it into the HTTP2 stream priority, and

- take a subscription-dependency if it has been provided and map the HTTP2 stream for the parent subscription into the HTTP2 stream dependency.

3.2. Subscription Multiplexing

It is possible that updates across subscriptions might be delivered in a different sequence than the encapsulated records were generated. Reasons for this might include (but are not limited to):

- generation of event records on different line cards

- replay of pushed information, and

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

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

Therefore each notification message will include a timestamp to provide a Receiver with its best information indicating when a particular record was generated. Use of this timestamp can give an indication of the state of objects at a Publisher. This is especially important when state-entangled information is received across different subscriptions. Note that use of notification message timestamps may not indicate a the exact time of occurrence. 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.

4. Encoded Subscription and Notification Message Examples

4.1. Restconf Subscription and Events over HTTP1.1

Subscribers can dynamically learn whether a RESTCONF server supports various types of Event or Yang datastore subscription capabilities. This is done by issuing an HTTP request OPTIONS, HEAD, or GET on the stream. Some examples building upon the Call flow for HTTP1.1 from Section 3.2.2 are:

```
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 as a 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-20150203] transport strategy to push filtered events from the event stream.

The Publisher MUST support individual parameters within the POST request body for 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
POST /restconf/operations/ietf-event-notifications:
     establish-subscription HTTP/1.1
      Host: example.com
      Content-Type: application/yang-data+json

      {
        "ietf-event-notifications:input" : {
          "stream": "push-data"
          "period" : 5,
          "xpath-filter" : "/ex: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 content 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>2015-03-09T19:14:56.233Z</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 [RFC7951]:

```
{
  "ietf-restconf:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56.233Z",
    "datastore-push:datastore-contents": {
      "example-mod:foo": { "bar": "some_string" }
    }
  }
}
```

To modify a Subscription, the subscriber issues another POST 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:

```
POST /restconf/operations/ietf-event-notifications:
      modify-subscription HTTP/1.1
      Host: example.com
      Content-Type: application/yang-data+json

      {
        "ietf-event-notifications:input" : {
          "subscription-id": 100,
          "period" : 10
        }
      }
```

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

4.2. Event Notification over HTTP2

The basic encoding will look as below. It will consists of a JSON representation wrapped in an HTTP2 header.

```
HyperText Transfer Protocol 2
      Stream: HEADERS, Stream ID: 5
      Header: :method: POST
      Stream: HEADERS, Stream ID: 5

{
  "ietf-yangpush:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56.233Z",
    "datastore-push:datastore-contents": {
      "foo": { "bar": "some_string" }
    }
  }
}
```

5. Security Considerations

Subscriptions could be used to intentionally or accidentally overload the resources of a Publisher. For this reason, it is important that the Publisher has the ability to prioritize the establishment and push of notification messages where there is the potential for resource exhaust. 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 of Configured Subscriptions could be used to overwhelm a Receiver which doesn't even support Subscriptions. There are two protections here. First, notification messages for Configured Subscriptions MUST only be transmittable over encrypted transports. Clients which do not want pushed content need only

terminate or refuse any transport sessions from the Publisher. Second, the HTTP transport augmentation on the Receiver must send an HTTP 200 OK to a subscription started notification before the Publisher starts streaming any subscribed content.

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, HTTP2 transport such as HTTP2) should be selected.

6. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, Michael Scharf, and Guangying Zheng.

7. References

7.1. 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>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <http://www.rfc-editor.org/info/rfc8040>.

[sn] Voit, E., Clemm, A., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to Event Notifications", February 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-subscribed-notifications/>.

7.2. Informative References

[GRPC] "RPC framework that runs over HTTP2", August 2017, <https://grpc.io/>.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <http://www.rfc-editor.org/info/rfc7923>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <http://www.rfc-editor.org/info/rfc7951>.

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <http://www.rfc-editor.org/info/rfc8071>.

[W3C-20150203] "Server-Sent Events, World Wide Web Consortium CR CR-eventsource-20121211", February 2015, <https://www.w3.org/TR/2015/REC-eventsource-20150203/>.

[yang-push] Clemm, A., Voit, E., Gonzalez Prieto, A., Prasad Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", March 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

Appendix A. 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.

A.1. Call Home

Implementations should include the ability to transparently incorporate 'call home' [RFC8071] so that secure TLS connections can originate from the desired device.

A.2. TLS Heartbeat

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 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 any Subscription related TCP sessions between those endpoints being torn down. The subscription can then attempt to re-establish.

Appendix B. RESTCONF over GRPC

An initial goal for this document was to support [GRPC] transport seamlessly without any mapping or extra layering. However there is an incompatibility of RESTCONF and GRPC. RESTCONF uses HTTP GET, and GRPC uses HTTP2's POST rather than GET. As GET is used across RESTCONF for things like capabilities exchange, a seamless mapping depends on specification changes outside the scope of this document. If/when GRPC supports GET, or RESTCONF is updated to support POST, this should be revisited. It is hoped that the resulting fix will be transparent to this document.

Appendix C. Changes between revisions

(To be removed by RFC editor prior to publication)

v01 - v03

- Terminoology aligned with draft-voit-netconf-notification-messages.
- Tweaks to wording/capitalization/format.

v01 - v02

- Removed sections now redundant with [sn] and [yang-push] such as: mechanisms for subscription maintenance, terminology definitions, stream discovery.
- 3rd party subscriptions are out-of-scope.
- SSE only used with Restconf and HTTP1.1 Dynamic Subscriptions
- Timeframes for event tagging are self-defined.
- Clean-up of wording, references to terminology, section numbers.

v00 - v01

- Removed the ability for more than one subscription to go to a single HTTP2 stream.
- Updated call flows. Extensively.
- SSE only used with Restconf and HTTP1.1 Dynamic Subscriptions
- HTTP is not used to determine that a Receiver has gone silent and is not Receiving Event Notifications
- Many clean-ups of wording and terminology

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Alberto Gonzalez Prieto
VMWare

Email: agonzalezpri@vmware.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com