

NETCONF  
Internet-Draft  
Intended status: Informational  
Expires: April 15, 2016

E. Voit  
A. Clemm  
A. Tripathy  
E. Nilsen-Nygaard  
A. Gonzalez Prieto  
Cisco Systems  
October 13, 2015

Restconf subscription and HTTP push for YANG datastores  
draft-voit-restconf-yang-push-00

## Abstract

This document defines Restconf subscription and push mechanisms to continuously stream information from YANG datastores over HTTP. These mechanisms allow client applications or operations support systems to request custom sets of updates from a YANG datastore. This document also specifies how to stream updates over HTTP without Restconf. In either case, updates are pushed by a datastore to a receiver per a subscription policy, without requiring continuous requests.

## 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 April 15, 2016.

## Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

Internet-Draft

Restconf Push

October 2015

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">2</a>
<a href="#">2.</a>	Terminology . . . . .	<a href="#">3</a>
<a href="#">3.</a>	Solution . . . . .	<a href="#">4</a>
<a href="#">3.1.</a>	Subscription Model . . . . .	<a href="#">4</a>
<a href="#">3.2.</a>	Subscription states at Publisher . . . . .	<a href="#">5</a>
3.3.	Mechanisms for Subscription Establishment and Maintenance	6
<a href="#">3.4.</a>	Negotiation of Subscription Policies . . . . .	<a href="#">8</a>
<a href="#">3.5.</a>	Support for Periodic and On-change . . . . .	<a href="#">8</a>
<a href="#">3.6.</a>	Filters and Streams . . . . .	<a href="#">9</a>
<a href="#">3.7.</a>	Authorization . . . . .	<a href="#">9</a>
<a href="#">3.8.</a>	Subscription Multiplexing . . . . .	<a href="#">9</a>
<a href="#">3.9.</a>	Push Data Stream and Transport Mapping . . . . .	<a href="#">10</a>
<a href="#">3.10.</a>	YANG Tree . . . . .	<a href="#">14</a>
<a href="#">4.</a>	YANG Module . . . . .	<a href="#">15</a>
<a href="#">5.</a>	Security Considerations . . . . .	<a href="#">18</a>
<a href="#">6.</a>	References . . . . .	<a href="#">18</a>
<a href="#">6.1.</a>	Normative References . . . . .	<a href="#">18</a>
<a href="#">6.2.</a>	Informative References . . . . .	<a href="#">19</a>
<a href="#">Appendix A.</a>	Dynamic YANG Subscription when the Subscriber and Receiver are different . . . . .	<a href="#">20</a>
<a href="#">Appendix B.</a>	End-to-End Deployment Guidance . . . . .	<a href="#">21</a>
<a href="#">B.1.</a>	Call Home . . . . .	<a href="#">21</a>
<a href="#">B.2.</a>	TLS Heartbeat . . . . .	<a href="#">21</a>
<a href="#">B.3.</a>	Putting it together . . . . .	<a href="#">22</a>
Authors' Addresses	. . . . .	<a href="#">22</a>

## [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 [\[netconf-yang-push\]](#). Restconf support of subscriptions, with HTTP transport of pushed updates is also needed by the market. This document provides such a specification.

Key benefits of pushing data via HTTP include:

- o Ability to configure static subscriptions on a Publisher

- o Ability for the Publisher to initiate communications with the Receiver
- o Ability of a Subscriber to be different from the Receiver

There are also additional benefits which can be realized when pushing updates via HTTP/2 [[RFC7540](#)]:

- o Subscription multiplexing over independent HTTP/2 streams
- o Stream prioritization
- o Stream dependencies
- o Flow control on independent streams
- o Header compression

These additional benefits will address issues resulting from head-of-line blocking and relative subscription priority.

To maximize transport independence of YANG subscription methods, this document reuses many capabilities of [[netconf-yang-push](#)][] including:

- o Operations for creating, modifying and deleting subscriptions
- o Syntax and parameters for negotiating the subscription
- o YANG data model to manage subscriptions
- o Mechanisms to communicate subscription filters and data 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 YANG Subscription:** Subscription negotiated with Publisher via create, modify, and delete control plane signaling messages.

**Publisher:** an entity responsible for distributing subscribed YANG object data per the terms of a Subscription. In general, a Publisher

Voit, et al.

Expires April 15, 2016

[Page 3]

---

Internet-Draft

Restconf Push

October 2015

is the owner of the YANG datastore that is subjected to the Subscription.

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

**Static YANG 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:** 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 mechanisms that allow subscribed information updates to be pushed from a YANG datastore. Subscriptions may either be initiated via requests by Subscribers, or statically configured on a Publisher. As in [[netconf-yang-push](#)], Publisher must respond to a subscription request explicitly positively or negatively. Negative responses will include information about why the Subscription was not accepted, in order to facilitate converging on an acceptable set of

Subscription parameters.

Once a Subscription has been established, updates are pushed to the Receiver until the Subscription terminates. Based on parameters within the Subscription, these updates can be streamed immediately as any subscribed objects change, or sent periodically.

### [3.1.](#) Subscription Model

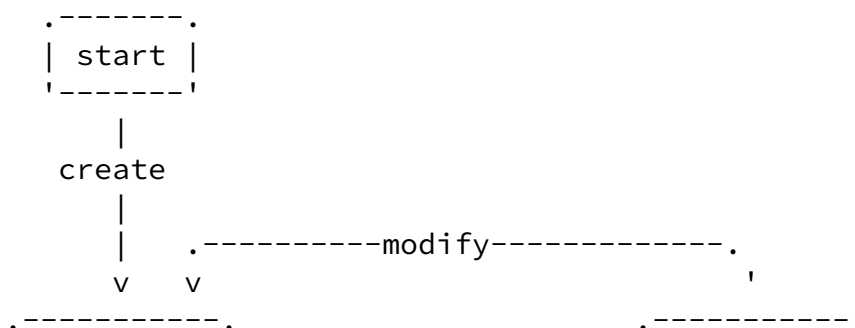
Subscriptions use the base data model from [[netconf-yang-push](#)]. This model is extended with several optional parameters for Subscription Priority and Subscription Dependency. These parameters allow a Subscriber or other configuration interface to assert how it prefers the Publisher allocate resources when handling multiple Subscriptions. These parameters are intended to be used in conjunction with the transport layer. Specifically, when a new Subscription is being established with an underlying transport is HTTP/2, these parameters may be directly mapped into HTTP/2 to

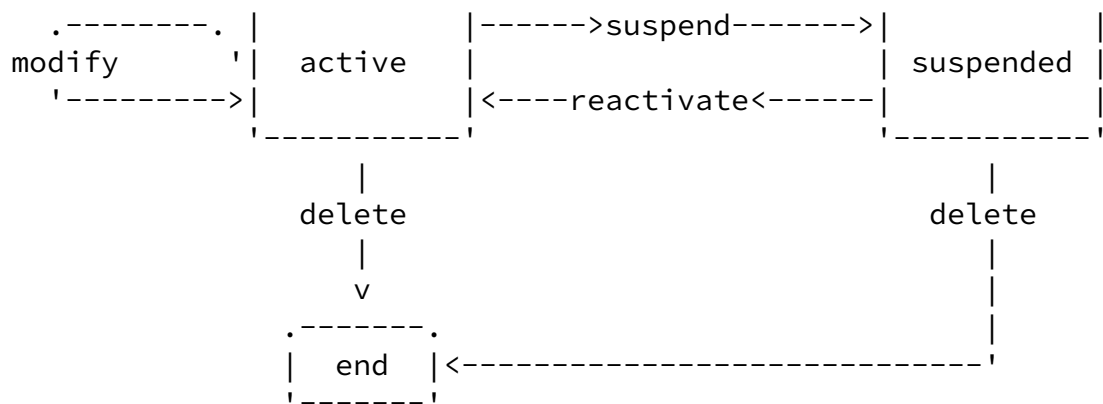
prioritize transport and to assist with flow control of individual streams.

### [3.2.](#) Subscription states at Publisher

Below is the state machine for the Publisher. It is important to note that a Subscription doesn't exist at the Publisher until it is accepted and made active. The assertion of a <create-subscription> by a Subscriber is insufficient for that asserted subscription to be externally visible via this state machine.

Subscription states at Publisher





Of interest in this state machine are the following:

- o Successful `<create-subscription>` or `<modify-subscription>` actions must put the subscription into an active state.
- o Failed `<modify-subscription>` actions will leave the subscription in its previous state, with no visible change to any streaming updates.
- o A `<delete-subscription>` action will delete the entire subscription.

### [3.3.](#) 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.

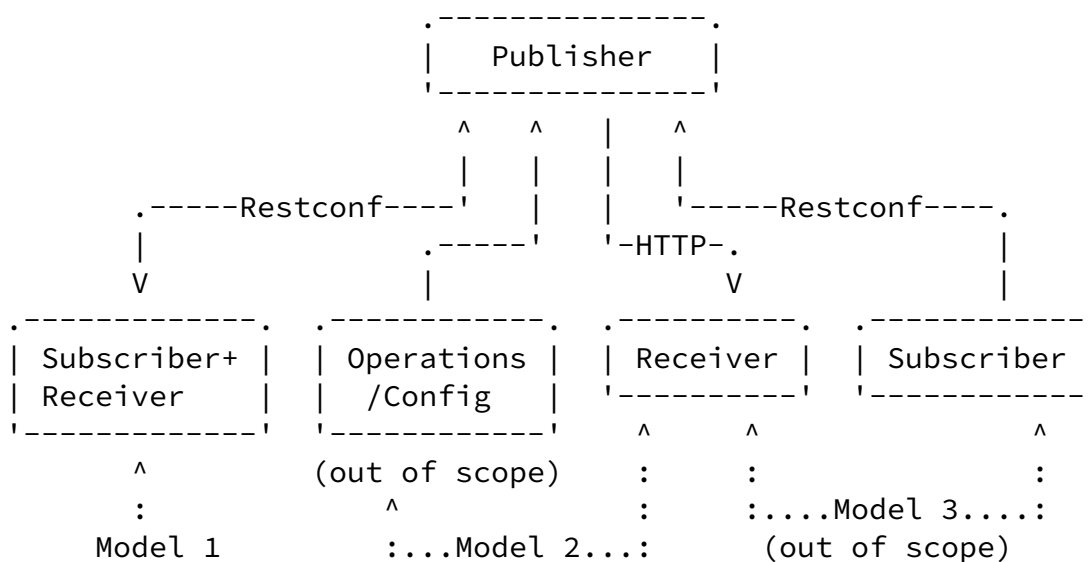
Dynamic YANG Subscriptions are signaled Subscriptions aimed at the running datastore and are unable to impact the startup configuration. They should always terminate when there is loss of transport session connectivity between the Publisher and Receiver.

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 YANG Subscription: Subscriber and Receiver are the same
2. Static YANG Subscription
3. Dynamic YANG 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. Figure 2 shows such a scenario.



### [3.3.1.](#) Dynamic YANG Subscription: Subscriber and Receiver are the same

With all Dynamic YANG Subscriptions, as with [\[netconf-yang-push\]](#) 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.

### [3.3.2.](#) Static YANG Subscription

With a Static YANG 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 <create-subscription> information identified in [section 3.3.1](#). This information also includes the Receiver address, encoding selection, and any 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.

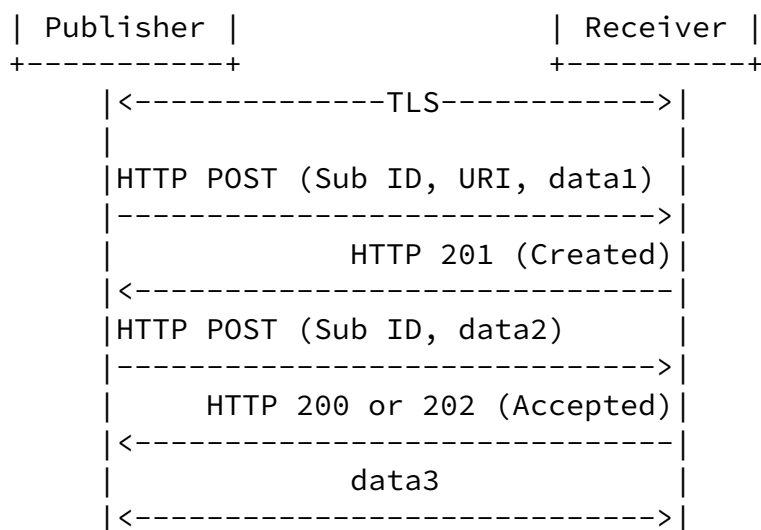
Post messages will be addressed to HTTP augmentation code on the Receiver capable accepting and responding to Subscription Updates. At least the initial Post message must include the URI for the subscribed resource. This URI can be retained for future use by the Receiver.

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.





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.

### 3.4. Negotiation of Subscription Policies

When using signaling to create a Dynamic YANG Subscription, negotiable parameters will include the same negotiable parameters defined within [[netconf-yang-push](#)].

Additionally, negotiation may also include Subscription Priority. A Publisher may accept a Subscriber asserted Priority, as well as rejecting a subscription with a hint at what priority might be accepted.

### 3.5. Support for Periodic and On-change

Implementations must support periodic and/or on-change subscriptions as defined in [[netconf-yang-push](#)].

### [3.6.](#) Filters and Streams

Implementations must support filters and streams as defined in [\[netconf-yang-push\]](#).

### [3.7.](#) Authorization

Same authorization model for data as [\[netconf-yang-push\]](#) will be used. This includes functions of the Netconf Access Control Model [\[RFC6536\]](#) applied to objects to be pushed via Restconf.

A Subscription (including a Static YANG Subscription) may only be established if the Subscriber or some entity statically configuring via the Publisher's operational interface has read access to the target data node.

### [3.8.](#) Subscription Multiplexing

When pushed directly over HTTP/2, it is expected that each Subscription Update will be allocated a separate 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, 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
- o 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.

### [3.9.](#) 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:

- o A global subscription ID correlator, referencing the name of the Subscription on whose behalf the notification is sent.
- o 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.
- o An event time which contains the time stamp at publisher when the event is generated.

#### [3.9.1.](#) 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, 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 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
```

### [3.9.2.](#) 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" }
    }
  }
}
```

```

    }
  }
}

```

### 3.10. YANG Tree

Below is the object tree for the model. All items are imported from [\[netconf-yang-push\]](#) except for the addition of "subscription-priority" and "subscription-dependency".

```

module: ietf-restconf-yang-push
+-ro system-streams
|  +-ro system-stream*                system-stream
+-rw filters
|  +-rw filter* [filter-id]
|      +-rw filter-id                filter-id
|      +-rw subtree-filter?          subtree-filter
|      +-rw xpath-filter?            yang:xpath1.0
+-rw subscription-config
|  +-rw datastore-push-subscription*  [subscription-id]
|  +---rw datastore-push-subscription* [subscription-id]
|      +---rw subscription-id          subscription-id
|      +---rw target-datastore?        datastore
|      +---rw stream?                  system-stream
|      +---rw encoding?                encoding
|      +---rw start-time?              yang:date-and-time
|      +---rw stop-time?               yang:date-and-time
|      +---rw (update-trigger)?
|          | +---:(periodic)
|          | | +---rw period?          yang:timeticks
|          | +---:(on-change)
|          | | +---rw no-synch-on-start? empty
|          | | +---rw dampening-period yang:timeticks
|          | | +---rw excluded-change*  change-type
|      +---rw (filterspec)?
|          | +---:(inline)
|          | | +---rw subtree-filter?  subtree-filter

```

```

|      | | +---rw xpath-filter?        yang:xpath1.0
|      | +---:(by-reference)
|      | +---rw filter-ref?            filter-ref
|      +---rw receiver-address

```



```

|         +-rw (push-base-transport)?
|         |   +--:(tcpudp)
|         |   +-rw tcpudp
|         |   +-rw address?          inet:host
|         |   +-rw port?             inet:port-number
|         +---rw subscription-priority?  uint8
|         +---rw subscription-dependency? string
+-ro subscriptions
  +---ro datastore-push-subscription* [subscription-id]
  +---ro subscription-id               subscription-id
  +---ro configured-subscription?      empty
  +---ro subscription-status?          identityref
  +---ro target-datastore?             datastore
  +---ro stream?                       system-stream
  +---ro encoding?                     encoding
  +---ro start-time?                   yang:date-and-time
  +---ro stop-time?                    yang:date-and-time
  +---ro (update-trigger)?
  |   +--:(periodic)
  |   |   +---ro period?                yang:timeticks
  |   +--:(on-change)
  |   |   +---ro no-synch-on-start?      empty
  |   |   +---ro dampening-period        yang:timeticks
  |   |   +---ro excluded-change*        change-type
  +---ro (filterspec)?
  |   +--:(inline)
  |   |   +---ro subtree-filter?         subtree-filter
  |   |   +---ro xpath-filter?          yang:xpath1.0
  |   +--:(by-reference)
  |   |   +---ro filter-ref?             filter-ref
  +---ro receiver-address
  |   +---ro (push-base-transport)?
  |   |   +--:(tcpudp)
  |   |   +---ro tcpudp
  |   |   +---ro address?                inet:host
  |   |   +---ro port?                   inet:port-number
  +---rw subscription-priority?         uint8
  +---rw subscription-dependency?       string

```

#### 4. YANG Module

```

namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-push";

prefix "rc-push";

```

```
import ietf-datastore-push {
  prefix ds-push;
}

organization
  "IETF NETCONF (Network Configuration) Working Group";

contact
  "WG Web:      <http://tools.ietf.org/wg/netconf/>
  WG List:      <mailto:netconf@ietf.org>

  WG Chair:     Mahesh Jethanandani
                <mailto:mjethanandani@gmail.com >

  WG Chair:     Mehmet Ersue
                <mailto:mehmet.ersue@nokia.com>

  Editor:       Eric Voit
                <mailto:evoit@cisco.com>

  Editor:       Alexander Clemm
                <mailto:alex@cisco.com>

  Editor:       Ambika Prasad Tripathy
                <mailto:ambtripa@cisco.com>

  Editor:       Einar Nilsen-Nygaard
                <mailto:einarnn@cisco.com>

  Editor:       Alberto Gonzalez Prieto
                <mailto:albertgo@cisco.com>";

description
  "This module contains conceptual YANG specifications for
  Restconf datastore push.";

revision 2015-10-01 {
  description
    "Initial revision.";
  reference "restconf YANG Datastore push";
}

grouping subscription-qos {
  description
    "This grouping describes Quality of Service information
    concerning a subscription. This information is passed to lower
    layers for transport prioritization and treatment";
```

leaf subscription-priority {

Internet-Draft

Restconf Push

October 2015

```
    type uint8;
    description
        "Relative priority for a subscription.  Allows an underlying
        transport layer perform informed load balance allocations
        between various subscriptions";
}
leaf subscription-dependency {
    type string;
    description
        "Provides the Subscription ID of a parent subscription
        without which this subscription should not exist. In
        other words, there is no reason to stream these objects
        if another subscription is missing.";
}
}

augment "/ds-push:subscription-config/" +
    "ds-push:datastore-push-subscription" {
    description
        "Aguments configured subscriptions with QoS parameters.";
    uses subscription-qos;
}

augment "/ds-push:subscriptions/" +
    "ds-push:datastore-push-subscription" {
    description
        "Aguments the list of currently active subscriptions
        with QoS parameters.";
    uses subscription-qos;
}

augment "/ds-push:create-subscription/" +
    "ds-push:input" {
    description
        "Aguments the create subscription rpc with QoS parameters.";
    uses subscription-qos;
}

augment "/ds-push:modify-subscription/" +
    "ds-push:input" {
```

```
        description
        "Aguments the modify subscription rpc with QoS parameters.";
        uses subscription-qos;
    }

}
```

## 5. 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.

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.

## 6. References

### 6.1. 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>>.

Voit, et al.

Expires April 15, 2016

[Page 18]

---

Internet-Draft

Restconf Push

October 2015

- [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", July 2015, <<https://tools.ietf.org/html/draft-ietf-netconf-call-home-09>>.

[netconf-yang-push]

Clemm, Alexander., Gonzalez Prieto, Alberto., and Eric. Voit, "Subscribing to YANG datastore push updates",

October 2015, <<https://datatracker.ietf.org/doc/draft-clemm-netconf-yang-push/>>.

[pub-sub-reqs]

Voit, Eric., Clemm, Alexander., and Alberto. Gonzalez Prieto, "Subscribing to datastore push updates", October 2015, <<https://datatracker.ietf.org/doc/draft-ietf-i2rs-pub-sub-requirements/>>.

[restconf]

Bierman, Andy., Bjorklund, Martin., and Kent. Watsen, "RESTCONF Protocol", July 2015, <<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>>.

Voit, et al.

Expires April 15, 2016

[Page 19]

---

Internet-Draft

Restconf Push

October 2015

[yang-json]

Lhotka, Ladislav., "JSON Encoding of Data Modeled with YANG", October 2015, <<https://datatracker.ietf.org/doc/draft-ietf-netmod-yang-json/>>.

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

The methods of Sections [3.3.1](#) and [3.3.2](#) 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.

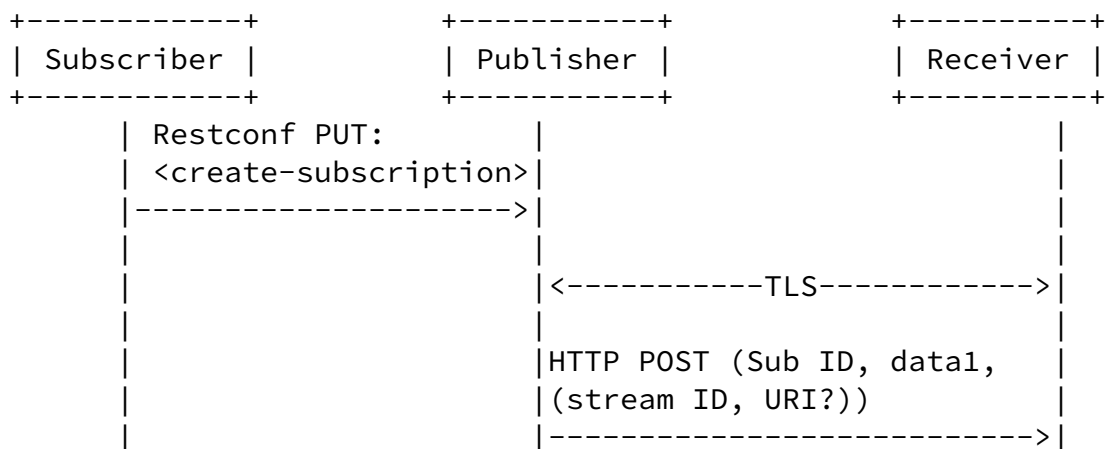
- o Restconf is not desired on the Receiver.
- o The Publisher doesn't want to maintain Restconf subscriptions with many Receivers.

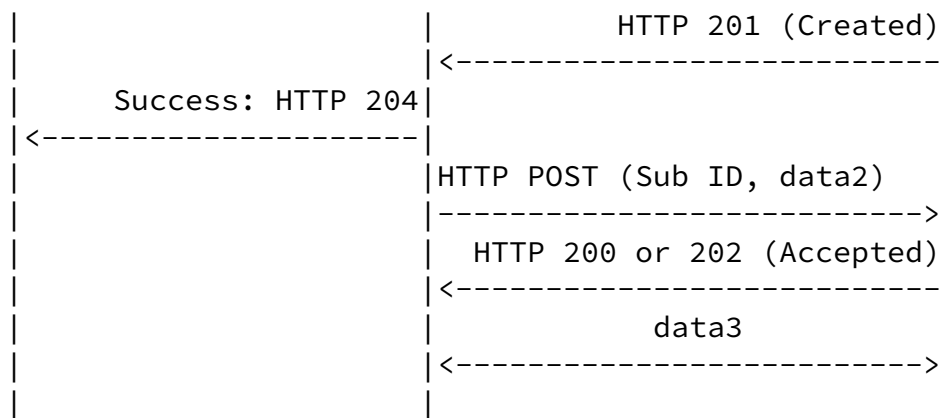
To do this, first the necessary information must be signaled as part of the <create-subscription>. This includes all the information described in [section 3.3.2](#), with the exception of the security credentials. (It is assumed that any security credentials required for establishing any transport connections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





## [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 as desired.

### [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](mailto:evoit@cisco.com)

Alexander Clemm  
Cisco Systems

Email: [alex@cisco.com](mailto:alex@cisco.com)

Ambika Prasad Tripathy  
Cisco Systems

Email: [ambtripa@cisco.com](mailto:ambtripa@cisco.com)

Einar Nilsen-Nygaard  
Cisco Systems

Email: [einarnn@cisco.com](mailto:einarnn@cisco.com)

Alberto Gonzalez Prieto  
Cisco Systems

Email: [albertgo@cisco.com](mailto:albertgo@cisco.com)