

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2020

G. Zheng
T. Zhou
Huawei
A. Clemm
Futurewei
T. Graf
Swisscom
P. Francois
INSA-Lyon
March 9, 2020

UDP based Publication Channel for Streaming Telemetry
draft-unyte-netconf-udp-pub-channel-00

Abstract

This document describes a UDP-based publication channel for streaming telemetry use to collect data from devices. A new shim header is proposed to facilitate the distributed data collection mechanism which directly pushes data from line cards to the collector. Because of the lightweight UDP encapsulation, higher frequency and better transit performance can be achieved.

Requirements Language

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](#)].

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 <https://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 September 10, 2020.

Copyright Notice

Copyright (c) 2020 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 (<https://trustee.ietf.org/license-info>) in effect on the date of 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

1.	Introduction	3
2.	Terminologies	4
3.	Transport Mechanisms	4
3.1.	Dynamic Subscription	4
3.2.	Configured Subscription	5
4.	UDP Transport for Publication Channel	6
4.1.	Design Overview	6
4.2.	Data Format of the UPC Message Header	7
4.3.	Options	8
4.3.1.	Fragmentation Option	8
4.4.	Data Encoding	9
5.	Using DTLS to Secure UPC	10
5.1.	Transport	10
5.2.	Port Assignment	11
5.3.	DTLS Session Initiation	11
5.4.	Sending Data	11
5.5.	Closure	12
6.	Congestion Control	12
7.	A YANG Data Model for Management of UPC	13
8.	YANG Module	13
9.	IANA Considerations	15
10.	Security Considerations	16
11.	Acknowledgements	16
12.	References	16
12.1.	Normative References	17
12.2.	Informative References	18
12.3.	URIs	19
	Authors' Addresses	19

1. Introduction

Streaming telemetry refers to sending a continuous stream of operational data from a device to a remote receiver. This provides an ability to monitor a network from remote and to provide network analytics. Devices generate telemetry data and push that data to a collector for further analysis. By streaming the data, much better performance, finer-grained sampling, monitoring accuracy, and bandwidth utilization can be achieved than with polling-based alternatives.

Sub-Notif [[RFC8639](#)] defines a mechanism that allows a collector to subscribe to updates of YANG-defined data that is maintained in a YANG [[RFC7950](#)] datastore. The mechanism separates the management and control of subscriptions from the transport that is used to actually stream and deliver the data. Three transports, NETCONF transport [[RFC8640](#)], RESTCONF transport [[I-D.ietf-netconf-restconf-notif](#)] and HTTPS transport [[I-D.ietf-netconf-https-notif](#)], have been defined so far for the notification messages.

While powerful in its features and general in its architecture, in its current form the mechanism needs to be extended to stream telemetry data at high velocity from devices that feature a distributed architecture. The transports that have been defined so far, NETCONF and HTTP, are ultimately based on TCP and lack the efficiency needed to stream data continuously at high velocity. A lighter-weight, more efficient transport, e.g. a transport based on UDP is needed.

- o Firstly, data collector will suffer a lot of TCP connections from, for example, many line cards equipped on different devices.
- o Secondly, as no connection state needs to be maintained, UDP encapsulation can be easily implemented by hardware which will further improve the performance.
- o Thirdly, because of the lightweight UDP encapsulation, higher frequency and better transit performance can be achieved, which is important for streaming telemetry.

This document specifies a higher-performance transport option for Sub-Notif that leverages UDP. Specifically, it facilitates the distributed data collection mechanism described in [[I-D.zhou-netconf-multi-stream-originators](#)]. In the case of data originating from multiple line cards, the centralized design requires data to be internally forwarded from those line cards to the push server, presumably on a main board, which then combines the individual data items into a single consolidated stream. The

centralized data collection mechanism can result in a performance bottleneck, especially when large amounts of data are involved. What is needed instead is the support for a distributed mechanism that allows to directly push multiple individual substreams, e.g. one from each line card, without needing to first pass them through an additional processing stage for internal consolidation, but still allowing those substreams to be managed and controlled via a single subscription. The proposed UDP based Publication Channel (UPC) natively supports the distributed data collection mechanism.

The transport described in this document can be used for transmitting notification messages over both IPv4 and IPv6 [[RFC8200](#)].

While this document will focus on the data publication channel, the subscription can be used in conjunction with the mechanism proposed in [[RFC8639](#)] with extensions [[I-D.zhou-netconf-multi-stream-originators](#)].

2. Terminologies

Streaming Telemetry: refers to sending a continuous stream of operational data from a device to a remote receiver. This provides an ability to monitor a network from remote and to provide network analytics.

3. Transport Mechanisms

For a complete pub-sub mechanism, this section will describe how the UPC is used to interact with the Subscription Channel relying on NETCONF or RESTCONF.

3.1. Dynamic Subscription

Dynamic subscriptions for Sub-Notif are configured and managed via signaling messages transported over NETCONF [[RFC6241](#)] or RESTCONF [[RFC8040](#)]. The Sub-Notif defined RPCs which are sent and responded via the Subscription Channel (a), between the Subscriber and the Subscription Server of the Publisher. In this case, only one Receiver is associated with the Subscriber. In the Publisher, there may be multiple data originators. Notification messages are pushed on separate channels (b), from different data originators to the Receiver.

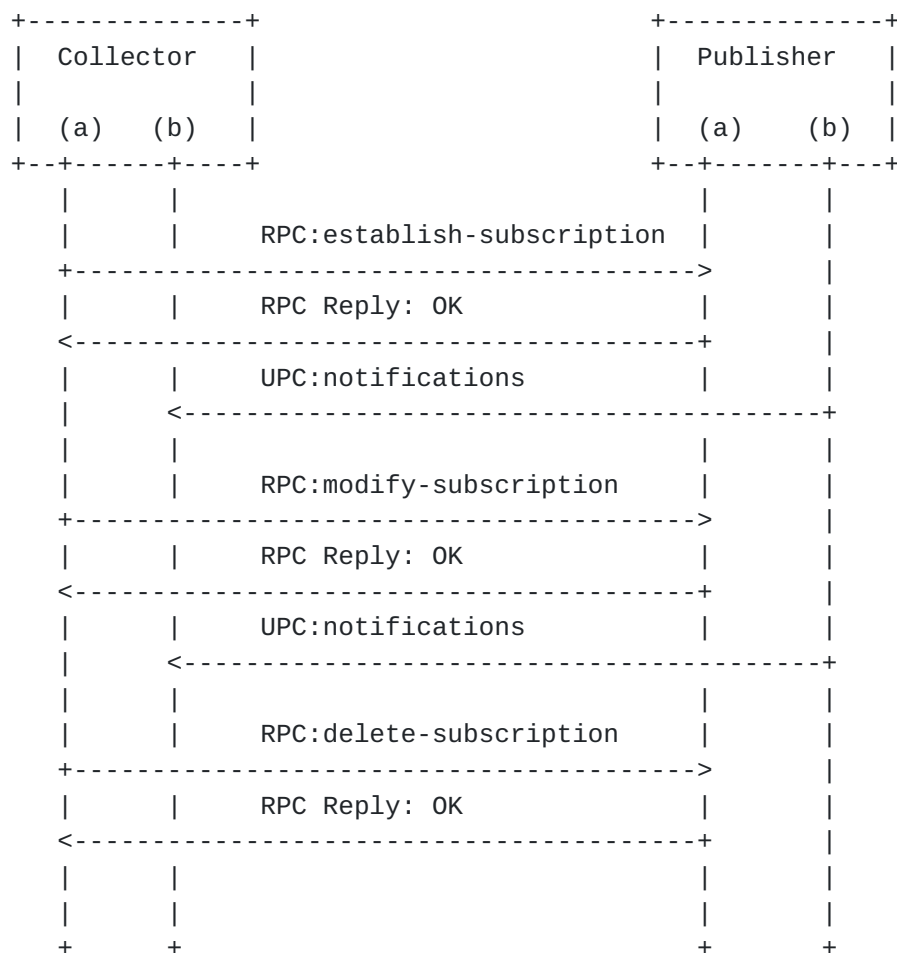


Fig. 2 Call Flow For Dynamic Subscription

In the case of dynamic subscription, the Receiver and the Subscriber SHOULD be colocated. So UPC can use the source IP address of the Subscription Channel as it's destination IP address. The Receiver MUST support listening messages at the IANA-assigned PORT-X or PORT-Y, but MAY be configured to listen at a different port.

For dynamic subscription, the Publication Channels MUST share fate with the subscription session. In other words, when the delete-subscription is received or the subscription session is broken, all the associated Publication Channels MUST be closed.

3.2. Configured Subscription

For a Configured Subscription, there is no guarantee that the Subscriber is currently in place with the associated Receiver(s). As defined in Sub-Notif, the subscription configuration contains the location information of all the receivers, including the IP address

and the port number. So that the data originator can actively send generated messages to the corresponding Receivers via the UPC.

The first message MUST be a separate subscription-started notification to indicate the Receiver that the pushing is started. Then, the notifications can be sent immediately without any wait.

All the subscription state notifications, as defined in [\[RFC8639\]](#), MUST be encapsulated to be separated notification messages.

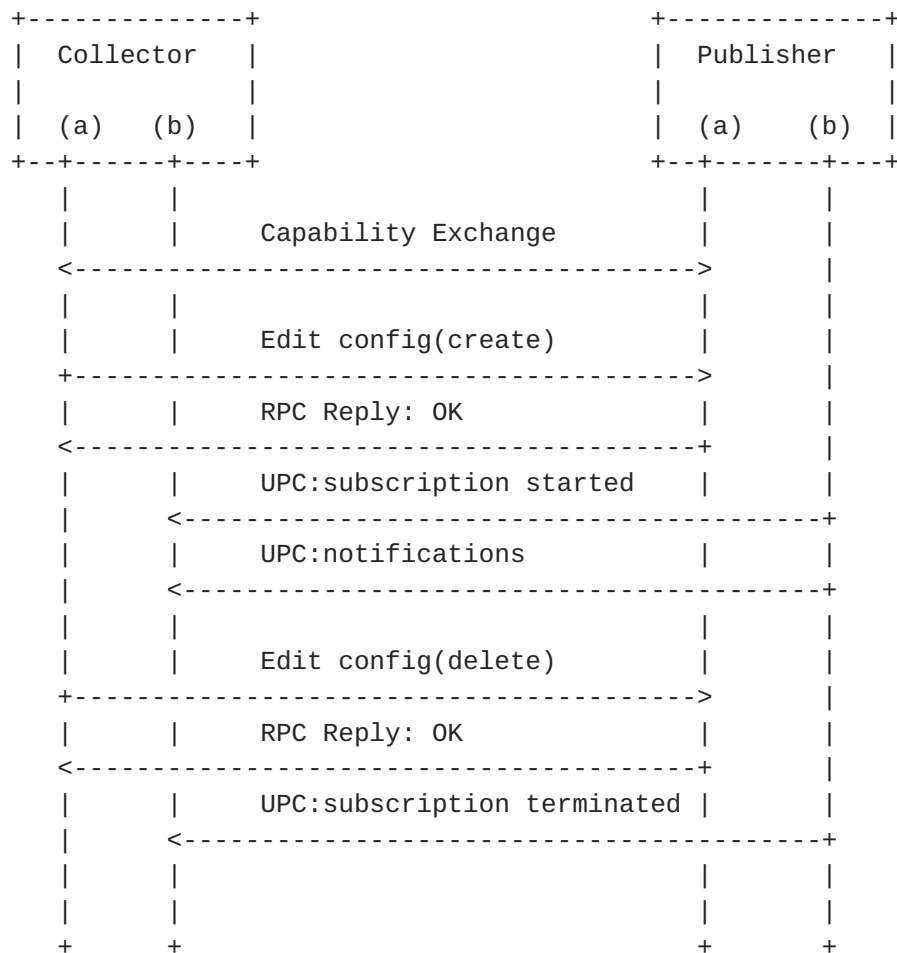


Fig. 3 Call Flow For Configured Subscription

4. UDP Transport for Publication Channel

4.1. Design Overview

As specified in Sub-Notif, the telemetry data is encapsulated in the NETCONF/RESTCONF notification message, which is then encapsulated and carried in the transport protocols, e.g. TLS, HTTP2. The following figure shows the overview of the typical UPC message structure.

- o The Message Header contains information that can facilitate the message transmission before de-serializing the notification message.
- o Notification Message is the encoded content that the publication channel transports. The common encoding method includes GPB [1], CBOR [RFC7049], JSON, and XML. [I-D.ietf-netconf-notification-messages] describes the structure of the Notification Message for both single notification and multiple bundled notifications.

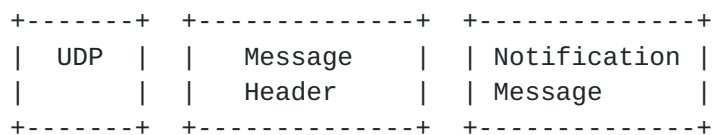


Fig. 4 UDP Publication Message Overview

4.2. Data Format of the UPC Message Header

The UPC Message Header contains information that can facilitate the message transmission before de-serializing the notification message. The data format is shown as follows.

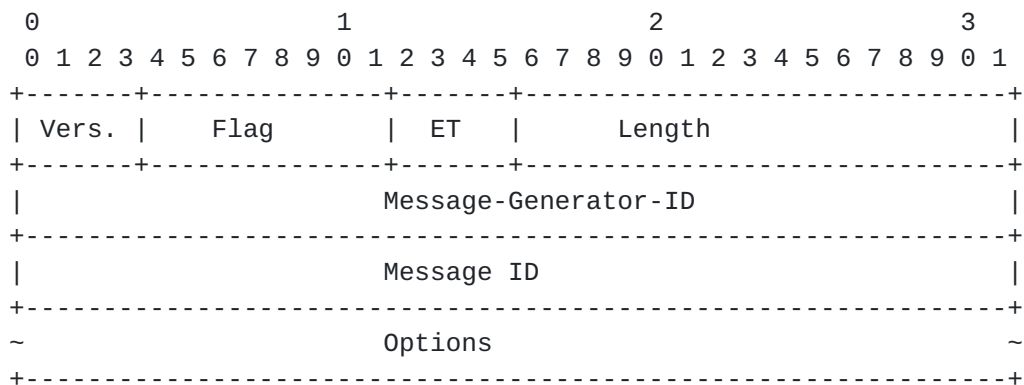


Fig. 3 UPC Message Header Format

The Message Header contains the following field:

- o Vers.: represents the PDU (Protocol Data Unit) encoding version. The initial version value is 0.
- o Flag: is a bitmap indicating what features this packet has and the corresponding options attached. Each bit associates to one feature and one option data. When the bit is set to 1, the associated feature is enabled and the option data is attached.

The sequence of the presence of the options follows the bit order of the bitmap. In this document, the flag is specified as follows:

- * bit 0, the fragmentation flag;
- * other bits are reserved.
- o ET: is a 4 bits identifier to indicate the encoding type used for the Notification Message. 16 types of encoding can be expressed:
 - * 0: GPB;
 - * 1: CBOR;
 - * 2: JSON;
 - * 3: XML;
 - * others are reserved.
- o Length: is the total length of the message, measured in octets, including message header.
- o Message-Generator-ID: is a 32-bit identifier of the process which created the notification message. This allows disambiguation of an information source, such as the identification of different line cards sending the notification messages. The source IP address of the UDP datagrams SHOULD NOT be interpreted as the identifier for the host that originated the UPC message. The entity sending the UPC message could be merely a relay.
- o The Message ID is generated continuously by the message generator. Different subscribers share the same notification ID sequence.
- o Options: is a variable-length field. The details of the Options will be described in the respective sections below.

4.3. Options

The order of packing the data fields in the Options field follows the bit order of the Flag field.

4.3.1. Fragmentation Option

UDP payload has a theoretical length limitation to 65535. Other encapsulation headers will make the actual payload even shorter. Binary encodings like GPB and CBOR can make the message compact. So

that the message can be encapsulated within one UDP packet, hence fragmentation will not easily happen. However, text encodings like JSON and XML can easily make the message exceed the UDP length limitation.

The Fragmentation Option can help not Application layer can split the YANG tree into several leaves. Or table into several rows. But the leaf or the row cannot be split any further. Now we consider a very long path. Since the GPB and CBOR are so compact, it's easy to fit into a UDP packet. But for JSON or XML, it is possible that even one leaf will exceed the UDP boundary.

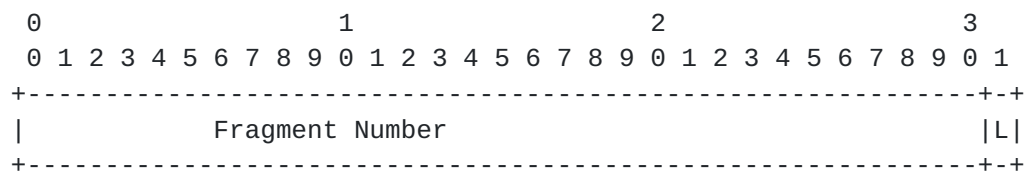


Fig. 5 Fragmentation Option Format

The Fragmentation Option is available in the message header when the fragmentation flag is set to 1. The option contains:

Fragment Number: indicates the sequence number of the current fragment.

L: is a flag to indicate whether the current fragment is the last one. When 0 is set, current fragment is not the last one, hence more fragments are expected. When 1 is set, current fragment is the last one.

[4.4.](#) Data Encoding

Subscribed data can be encoded in GPB, CBOR, XML or JSON format. It is conceivable that additional encodings may be supported as options in the future. This can be accomplished by augmenting the subscription data model with additional identity statements used to refer to requested encodings.

Implementation may support different encoding method per subscription. When bundled notifications is supported between the publisher and the receiver, only subscribed notifications with the same encoding can be bundled as one message.

5. Using DTLS to Secure UPC

The Datagram Transport Layer Security (DTLS) protocol [[RFC6347](#)] is designed to meet the requirements of applications that need secure datagram transport.

DTLS can be used as a secure transport to counter all the primary threats to UDP based Publication Channel:

- o Confidentiality to counter disclosure of the message contents.
- o Integrity checking to counter modifications to a message on a hop-by-hop basis.
- o Server or mutual authentication to counter masquerade.

In addition, DTLS also provides:

- o A cookie exchange mechanism during handshake to counter Denial of Service attacks.
- o A sequence number in the header to counter replay attacks.

5.1. Transport

As shown in Figure 6, the DTLS is layered next to the UDP transport is to provide reusable security and authentication functions over UDP. No DTLS extension is required to enable UPC messages over DTLS.

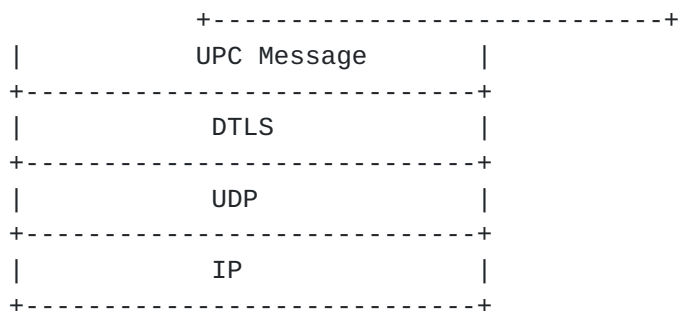


Fig. 6: Protocol Stack for DTLS secured UPC

The application implementer will map a unique combination of the remote address, remote port number, local address, and local port number to a session.

Each UPC message is delivered by the DTLS record protocol, which assigns a sequence number to each DTLS record. Although the DTLS implementer may adopt a queue mechanism to resolve reordering, it may

not assure that all the messages are delivered in order when mapping on the UDP transport.

Since UDP is an unreliable transport, with DTLS, an originator or relay may not realize that a collector has gone down or lost its DTLS connection state, so messages may be lost.

The DTLS record has its own sequence number, the encryption and decryption will be done by DTLS layer, UPC Message layer will not concern this.

5.2. Port Assignment

The Publisher is always a DTLS client, and the Receiver is always a DTLS server. The Receivers MUST support accepting UPC Messages on the UDP port PORT-Y, but MAY be configurable to listen on a different port. The Publisher MUST support sending UPC messages to the UDP port PORT-Y, but MAY be configurable to send messages to a different port. The Publisher MAY use any source UDP port for transmitting messages.

5.3. DTLS Session Initiation

The Publisher initiates a DTLS connection by sending a DTLS Client Hello to the Receiver. Implementations MUST support the denial of service countermeasures defined by DTLS. When these countermeasures are used, the Receiver responds with a DTLS Hello Verify Request containing a cookie. The Publisher responds with a DTLS Client Hello containing the received cookie, which initiates the DTLS handshake. The Publisher MUST NOT send any UPC messages before the DTLS handshake has successfully completed.

Implementations MUST support DTLS 1.0 [[RFC4347](#)] and MUST support the mandatory to implement cipher suite, which is TLS_RSA_WITH_AES_128_CBC_SHA [[RFC5246](#)] as specified in DTLS 1.0. If additional cipher suites are supported, then implementations MUST NOT negotiate a cipher suite that employs NULL integrity or authentication algorithms.

Where privacy is REQUIRED, then implementations must either negotiate a cipher suite that employs a non-NUL encryption algorithm or else achieve privacy by other means, such as a physically secured network.

5.4. Sending Data

All UPC messages MUST be sent as DTLS "application_data". It is possible that multiple UPC messages be contained in one DTLS record, or that a publication message be transferred in multiple DTLS

records. The application data is defined with the following ABNF [[RFC5234](#)] expression:

APPLICATION-DATA = 1*UPC-FRAME

UPC-FRAME = MSG-LEN SP UPC-MSG

MSG-LEN = NONZERO-DIGIT *DIGIT

SP = %d32

NONZERO-DIGIT = %d49-57

DIGIT = %d48 / NONZERO-DIGIT

UPC-MSG is defined in [section 5.2](#).

5.5. Closure

A Publisher MUST close the associated DTLS connection if the connection is not expected to deliver any UPC Messages later. It MUST send a DTLS close_notify alert before closing the connection. A Publisher (DTLS client) MAY choose to not wait for the Receiver's close_notify alert and simply close the DTLS connection. Once the Receiver gets a close_notify from the Publisher, it MUST reply with a close_notify.

When no data is received from a DTLS connection for a long time (where the application decides what "long" means), Receiver MAY close the connection. The Receiver (DTLS server) MUST attempt to initiate an exchange of close_notify alerts with the Publisher before closing the connection. Receivers that are unprepared to receive any more data MAY close the connection after sending the close_notify alert.

Although closure alerts are a component of TLS and so of DTLS, they, like all alerts, are not retransmitted by DTLS and so may be lost over an unreliable network.

6. Congestion Control

Congestion control mechanisms that respond to congestion by reducing traffic rates and establish a degree of fairness between flows that share the same path are vital to the stable operation of the Internet [[RFC2914](#)]. While efficient, UDP has no build-in congestion control mechanism. Because streaming telemetry can generate unlimited amounts of data, transferring this data over UDP is generally problematic. It is not recommended to use the UDP based publication channel over congestion-sensitive network paths. The only

environments where the UDP based publication channel MAY be used are managed networks. The deployments require the network path has been explicitly provisioned for the UDP based publication channel through traffic engineering mechanisms, such as rate limiting or capacity reservations.

7. A YANG Data Model for Management of UPC

The YANG model defined in [Section 9](#) has two leafs augmented into one place of Sub-Notif [[RFC8639](#)], plus one identities.

```
module: ietf-upc-subscribed-notifications
  augment /sn:subscriptions/sn:subscription/sn:receivers/sn:receiver:
    +--rw address?    inet:ip-address
    +--rw port?       inet:port-number
```

8. YANG Module

<CODE BEGINS> file "ietf-upc-subscribed-notifications@2020-02-26.yang"

```
module ietf-upc-subscribed-notifications {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-upc-subscribed-notifications";
  prefix upcsn;
  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }
  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  organization "IETF NETCONF (Network Configuration) Working Group";
  contact
    "WG Web:    <http://tools.ietf.org/wg/netconf/>
    WG List:    <mailto:netconf@ietf.org>

    Editor:     Guangying Zheng
                <mailto:zhengguangying@huawei.com>

    Editor:     Tianran Zhou
                <mailto:zhoutianran@huawei.com>

    Editor:     Alexander Clemm
                <mailto:alexander.clemm@huawei.com>";
```


description

"Defines UDP Publish Channel as a supported transport for subscribed event notifications.

Copyright (c) 2018 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in [Section 4.c](#) of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

revision 2020-02-26 {

description

"Initial version";

reference

"RFC XXXX: UDP based Publication Channel for Streaming Telemetry";

}

identity upc {

base sn:transport;

description

"UPC is used as transport for notification messages and state change notifications.";

}

identity encode-cbor {

base sn:encoding;

description

"Encode data using CBOR as described in [RFC 7049](#).";

reference

"[RFC 7049](#): Concise Binary Object Representation";

}

identity encode-gpb {

base sn:encoding;

description

"Encode data using gpb.";

}

grouping target-receiver {

description

"Provides a reusable description of a UPC target receiver.";


```
leaf address {
  type inet:ip-address;
  description
    "IP address of target upc receiver, which can be IPv4 address or
    IPV6 address.";
}
leaf port {
  type inet:port-number;
  description
    "Port number of target UPC receiver, if not specify, system
    should use default port number.";
}

leaf enable-fragmentation {
  type boolean;
  default false;
  description
    "The switch for the fragmentation feature. When disabled, the
    publisher will not allow fragmentation for a very large data";
}
}

augment "/sn:subscriptions/sn:subscription/sn:receivers/sn:receiver" {
  description
    "This augmentation allows UPC specific parameters to be
    exposed for a subscription.";
  uses target-receiver;
}
}
<CODE ENDS>
```

9. IANA Considerations

This RFC requests that IANA assigns three UDP port numbers in the "Registered Port Numbers" range with the service names "upc" and "upc-dtls". These ports will be the default ports for the UDP based Publication Channel for NETCONF and RESTCONF. Below is the registration template following the rules in [\[RFC6335\]](#).

Service Name: upc

Transport Protocol(s): UDP

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: UDP based Publication Channel

Reference: RFC XXXX

Port Number: PORT-X

Service Name: upc-dtls

Transport Protocol(s): UDP

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: UDP based Publication Channel (DTLS)

Reference: RFC XXXX

Port Number: PORT-Y

IANA is requested to assign a new URI from the IETF XML Registry [[RFC3688](#)]. The following URI is suggested:

URI: urn:ietf:params:xml:ns:yang:ietf-upc-subscribed-notifications

Registrant Contact: The IESG.

XML: N/A; the requested URI is an XML namespace.

This document also requests a new YANG module name in the YANG Module Names registry [[RFC7950](#)] with the following suggestion:

name: ietf-upc-subscribed-notifications

namespace: urn:ietf:params:xml:ns:yang:ietf-upc-subscribed-notifications

prefix: upcsn

reference: RFC XXXX

[10.](#) Security Considerations

TBD

[11.](#) Acknowledgements

The authors of this documents would like to thank Eric Voit, Tim Jenkins, and Huiyang Yang for the initial comments.

[12.](#) References

12.1. 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, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", [BCP 41](#), [RFC 2914](#), DOI 10.17487/RFC2914, September 2000, <<https://www.rfc-editor.org/info/rfc2914>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", [BCP 81](#), [RFC 3688](#), DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", [RFC 4347](#), DOI 10.17487/RFC4347, April 2006, <<https://www.rfc-editor.org/info/rfc4347>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, [RFC 5234](#), DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", [RFC 6241](#), DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", [BCP 165](#), [RFC 6335](#), DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", [RFC 6347](#), DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", [RFC 7049](#), DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", [RFC 7950](#), DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", [RFC 8040](#), DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, [RFC 8200](#), DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", [RFC 8639](#), DOI 10.17487/RFC8639, September 2019, <<https://www.rfc-editor.org/info/rfc8639>>.
- [RFC8640] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Dynamic Subscription to YANG Events and Datastores over NETCONF", [RFC 8640](#), DOI 10.17487/RFC8640, September 2019, <<https://www.rfc-editor.org/info/rfc8640>>.

12.2. Informative References

- [I-D.ietf-netconf-https-notif]
Jethanandani, M. and K. Watsen, "An HTTPS-based Transport for Configured Subscriptions", [draft-ietf-netconf-https-notif-01](#) (work in progress), October 2019.
- [I-D.ietf-netconf-notification-messages]
Voit, E., Jenkins, T., Birkholz, H., Bierman, A., and A. Clemm, "Notification Message Headers and Bundles", [draft-ietf-netconf-notification-messages-08](#) (work in progress), November 2019.
- [I-D.ietf-netconf-restconf-notif]
Voit, E., Rahman, R., Nilsen-Nygaard, E., Clemm, A., and A. Bierman, "Dynamic subscription to YANG Events and Datastores over RESTCONF", [draft-ietf-netconf-restconf-notif-15](#) (work in progress), June 2019.
- [I-D.zhou-netconf-multi-stream-originators]
Zhou, T., Zheng, G., Voit, E., and A. Clemm, "Subscription to Multiple Stream Originators", [draft-zhou-netconf-multi-stream-originators-10](#) (work in progress), November 2019.

12.3. URIs

[1] <https://developers.google.com/protocol-buffers/>

Authors' Addresses

Guangying Zheng
Huawei
101 Yu-Hua-Tai Software Road
Nanjing, Jiangsu
China

Email: zhengguangying@huawei.com

Tianran Zhou
Huawei
156 Beiqing Rd., Haidian District
Beijing
China

Email: zhoutianran@huawei.com

Alexander Clemm
Futurewei
2330 Central Expressway
Santa Clara, California
USA

Email: alex@futurewei.com

Thomas Graf
Swisscom
Binzring 17
Zuerich 8045
Switzerland

Email: thomas.graf@swisscom.com

Pierre Francois
INSA-Lyon
Lyon
France

Email: pierre.francois@insa-lyon.fr

