I2RS working group
Internet-Draft
Intended status: Standards Track
Expires: April 3, 2016

S. Hares
Huawei
A. Beirman
K. Watsen
Juniper
October 1, 2015

I2RS protocol strawman
draft-ietf-i2rs-protocol-strawman-00.txt

Abstract

This document provides a strawman proposal for the I2RS protocol covering the ephemeral data store. It provides Yang ephemeral statement, netconf protocol extensions for the ephemeral data store, and RESTCONF protocol extensions for the protocol data store.

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 3, 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 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

This documents is a strawman for the I2RS Protocol from early I2RS design team discusses. It focuses on the protocol extensions for ephemeral data store.

This draft provides suggests the following additions to support the I2RS ephemeral state:

- Yang ephemeral statement,

- NETCONF ([RFC6241]) protocol extensions for the ephemeral data store,

- RESTCONF ([I-D.ietf-netconf-restconf]) protocol extensions for the ephemeral data store

draft-hares-i2rs-protocol-strawman-examples provides provides examples of this strawman protocol use for I2RS. This draft uses a simple thermostat model to illustrate commands.

This draft is input to a NETCONF review and design team.

2. Resolve before publishing draft

1. (dean)Where will be the ephemeral datastore defined? I'm hearing discussions about ephemeral dat store is several places and it doesn't sound people have a common understanding of it. As side commend, I agree what you wrote down as high level layout on it.

   * (Andy)There does seem to be some overlap right now with opstate, wrt/ defining new datastores. To me, a datastore is just a way to refer to "data in the same state" within a protocol. To others, it is seen as a more concrete implementation requirement. The IETF will have to work this out.

2. (dean)What edits are allowed in the ephemeral data store. Should those be syntactically correct or syntacticly and semantically?

   * (Andy)This is the $64 question. Jeff has described at least one scenario where priority between 2 clients does not clearly solve edit contention. (Forget all the details but involves an entry that cannot be deleted because the result would leave an unresolved reference somewhere else in the data). There are 3 possible outcomes for a valid edit:

     1. no collisions; must be accepted

     2. partial overlap with better priority data

     3. complete overlap with better priority data

     Depending on stop-on-error or continue-on-error (2) will be accepted or not. This is where Jan (and I) start to worry about the client trying to be too clever, but Joel thinks a client could recover and deal with outcome (2).

- (andy) Clearly the data has to pass "field validation" (pass the typedef checks; can’t send int32=fred)

- Validation slows things down a lot, so datastore validation needs to be considered carefully. This is where I think routing expertise will help decide how much validation can really be skipped for a particular use-case.

3. (Andy)Here is an example of a routing use-case that is a challenge.

   - How is client priority used to make sure that a lessor client cannot insert a route with a shorter prefix into a RIB than an existing entry by a higher priority client?

   - This goes back to early questions that were never answered, such as "what exactly is an overlap/edit collision". It seems that some data models have to be written with client priority support in them, rather than something than can always be resolved by comparing exact instances in 2 client panes.

   - Examples of 2 clients trying to insert routes into the same RIB would be useful for the draft. I think we need that to explain all the things we mean by "overlap" when evaluating client panes.

4. (Anu)On priority: So the priority maximum will be same as the max-clients , if we are assigning unique priorities from 1 - max-clilents. (Eg , if max clients is 100 (leaf max clients , range 1 .. max) then priority range is also from 1 - max) So the leaf max-clients { .. range 1- 32 } represents priority information also , so it can be max-clients or max-priorities ??

   1. (Andy)The term ’max’ in YANG resolves to 4B-1 in this range, not 32. Actually, the text I sent allows for multiple clients to all have the default priority which is not good -- if client-id is needed to resolve collisions then there is no point to requiring a unique priority per client.

   2. (Andy)The priority is not required to be densely numbered. Whether there are 1 pane per client or 1 pane per priority or 1 giant blob full of everything, the code will be the same. The goal of "unique priority" is to require that only priority be saved in the meta-data for the ephemeral datastore. Without that, client-id and priority must be saved (per data node).

3. Definitions Related to Ephemeral Configuration

Currently the configuration systems managed by NETCONF ([RFC6241]) or RESTCONF ([I-D.ietf-netconf-restconf]) has three types of configuration: candidate, running, and startup running under the config=true flag.

- The candidate receives configuration changes from NETCONF/ RESTCONF.

- The running configuration is the configuration currently operating on a devices

- The start-up configuration is the configuration that survives a reboot.

The config=false flag has operational data which exists alongside the config=true data. However, at this point there is no datastored defined for configuration false.

operational

```
 ...........       ...........     ...........
 :Candidate : --> : running : --> :start-up  :
 ...........       ...........     ...........

 config true
 ---------------------------------------------
 config false
            ===============
            | operational  |
            | data         |
            ================
```

Figure 1

In reality, the running configuration becomes the intended configuration that is intended to be loaded into a device. The loading process of the intended configuation into a devices compares it against the actual devices and creates the actual configuration loaded into a box.

Some people denote the actual configuration as applied configuration. The [I-D.openconfig-netmod-opstate] denotes the actual configuration as derived state. This document will use the term actual configuration.

```
    ...........      ...........     ...........
    :Candidate : --> : running : --> :start-up :
    ...........      ...........     ...........

                    =============
                    | Intended  |
                    | config    |
                    =============
    config true
    ---------------------------------------------
    config false
                    =============
                    | Actual    |
                    | config    |
                    =============
                ______________
               | operational  |
               | data         |
               |______________|
```

Figure 2

Recently the [I-D.openconfig-netmod-opstate] has proposed that intended configuration, actual configuration, and the traditional type of operational data included as operational state. Operational data may include:

- derived state (e.g. negotiated bgp hold timer)

- operational state for counters or statistics (interface counters)

Again, this document will use the definitions above to discuss ephemeral state until the NETCONF WG agrees upon the changes to the state diagrams.

4. Definition of ephemeral datastore for NETCONF/RESTCONF

This section describes the properties of the ephemeral datastore. This approach to the ephemeral datastore is a panes-of-glass model.

The ephemeral data store has the following qualities:

1. The ephemeral datastore is a datastore holds configuration that is intended to not survive a reboot.

2. The ephemeral datastore is never locked.

3. The ephemeral datastore treated as N client panes where

   - the netconf/restconf server picks how many clients it supports

   - multi-head support is optional since max-clients allowed to be 1

4. Each client has a unique priority (see figure 3 for example yang statements)

   - If a client is not present in the i2rs-client list, then the worst priority value is assigned.

   - The best possible priority needs to be reserved for the system, or the protocol has to make a special case of system-set data

5. Each client writes into its own pane so there is no conflict within a pane. The implementation combines the panes into the appropriate image.

   - The difference between panes of glass is what the server retains from a partial or failed edit (due to conflicts in the panes (?editor))

   - It should be a valid operation to save nothing or to save all information (caching) within a pane of glass

6. A Partial operation is one where a subset of the written data is not applied because of better priority for that node. A partial operation is only allowed if the error-option is stop-on-error or continue-on-error.

   - stop-on-error - means that the configuration process stops when a write to the configuration detects an error due to write conflict.

   - continue-on-error - means the configuration process continues when a write to the configuration detects an error due to write process, and error reports are transmitted back to the client writing the error.

   - all-or-nothing - means that all of the configuration process is correctly applied or no configuration process is applied.

   - NETCONF stop-on-error and continue-on-error are not going to work. There is no mandated processing order for edits. For the stop-on-error and the continue-on-error process to work, the I2RS protocol extensions to NETCONF will have to force some processing order in order to support partial edits.

   - NETCONF has no current mechanism for reporting which edits were accepted and which edits were reject for partial operations. The I2RS protocol extensions will have to provide new error handling to the response data.

   - These features were removed from NETCONF (RFC 6241) because it was too complicated, and no company had implemented these features.

   - Interoperability issues must be considered in all three cases: a) all-or-nothing, b) stop-on-error, and c) continue-on-error.

7. caching is optional and and a server may retain the pain for each client.

   - If caching is not supported then the pane-of-glass never contains unaccepted data. Therefore, the server will return an error and will not retain the edit that caused the error.

   - If caching is supported, then the data is retained in the pane-of-glass, Therefore, if the higher priority data is removed then the lower priority data can be added. Notifications will be provided when this occurs. (?Editor)

```
      container i2rs-clients {
          leaf max-clients {
             config false;
             mandatory true;
             type uint32 {
               range "1 .. max";
             }
          }
          list i2rs-client {
             key name;
             unique priority;
             leaf name { ... }
             leaf priority { ... }
          }
       }
```

Figure 3

The ephemeral data store

```
    ...........      ...........     ...........
    :Candidate : --> : running : --> :start-up :
    ...........      ...........     ...........

                     ephemeral datastore
                 ...............
                 . ’’’’’’’’’’’ .   ===========
                 . ’ephemeral’ .-->|Intented |
                 . ’ config  ’ .   |Config   |
                 . ’’’’’’’’’’’ .   ===========
    config true  .             .
    -------------.-------------.-----------------
    config false . ’’’’’’’’’’’ .
                 . ’ephemeral’ .   ===========
                 . ’  actual ’ .-->| Actual  |
                 . ’  config ’ .   | config  |
                 . ’’’’’’’’’’’ .   ==========
                 ...............       ^
                ______________         |
               | operational  |        |
               | data         |--------|
               |______________|
```

Figure 4

5. Simple Thermostat Model

In this discussion of ephemeral configuration, this draft utilizes a simple thermostat model with the yang configuration found in figure 4.

```
   module thermostat {
     ..
     leaf desired-temp {
        type int32;
            units "degrees Celsius";
            description "The desired temperature";
            }

     leaf actual-temp {
        type int32;
            config false;
            units "degrees Celsius";
            description "The measured temperature";
            }
     }
```

Figure 4 - Simple thermostat model yabng

Figure 5 shows the diagram of the configuration state with the Simple thermostat model being attached to by an I2RS scheduler client receiving query information regarding intended configuration and actual configuration. Scheduler has a schedule set of temperatures to put in the thermostat.

```
    ...........      ...............   ...........
    :Candidate : --> : Desired temp:-->:start-up :
    ...........      ...............   ...........
                          |
                          V
                    ============    ===========
                    | Intended |----| I2RS    |
                    | config   |    |scheduler|
                    |          |    | client  |
                    ============    ===========
    config true                          ^
    -------------------------------      |
    config false                         |
                    =============        |
                    | Actual    |--------|
                    | config    |
                    =============
                        ^
                        |
                        |
                ______________
               | actual temp  |
               ----------------
```

Figure 5 - Scheduler client only

Figure 6 shows two I2RS clients talking to this model: scheduler and hold-temp. Scheduler has a schedule set of temperatures to put in the thermostat. Hold-temp holds the temperature at the same value. The hold-temp I2RS client has a higher priority than the scheduler client.

```
    ...........      ...............   ...........
    :Candidate :---: Desired temp   : -- :start-up :
    ...........      ...............   ...........
                          |
                          |          =============
                          |          |I2rs Client|
                          |          |scheduler  |
                          V         / ============
             ...................../
    ephemeral    . '''''''''''''''/.  ==============
    datastore    . 'desired-temp'---- |I2RS Client |
                 . '''''''''|'''' .   | hold temp  |
                 .          |     .   ==============
                 .          |     .   ============
                 .          |---------| intended |
                 .                 .  | config   |
                 .                 .  ============
    config true  .                 .
    -------------.-----------------.------
    config false .                 .
                 ...................
                    =============
                    | Actual    |
                    | config    |
                    =============
                        ^
                        |
                        |
                ______________
               | actual temp  |
               ----------------
```

Figure 6 - Two I2RS clients

Figure 7 shows a diagnostic test button within the thermostat system which tests the overheating response by altering the value of actual-temp. (This manual button is similar in concept to a manual button that puts an routing interface online or offline.)

```
    ...........      ...............   ...........
    :Candidate :---: Desired temp  : -- :start-up :
    ...........      ...............   ...........
                          |
                          |          =============
                          |          |I2rs Client|
                          |          |scheduler  |
                          V         / ============
                 ................../
    ephemeral    . ’’’’’’’’’’’’’’’/.   ==============
    datastore    . ’desired-temp’---- |I2RS Client |
                 . ’’’’’’’’’|’’’’ .   | hold temp  |
                 .          |     .   ==============
                 .          |     .   ============
                 .          |---------| intended |
                 .                .   | config   |
                 .                .   ============
    config true  .                .
    -------------.----------------.----------------
                 . ’’’’’’’’’’’’’’ .    ============
      ephemeral  . ’actual-temp ’------| actual   |
                 . ’’’’’’|’’’’’’\’.    | config   |
                 .       |       \     ============
    config false ........|........\
                                   \  ============
               ----------------     --| Diag-temp|
               | actual temp  |       ============
               ----------------
```

Figure 7 - Two I2RS clients

6. Yang changes

Yang needs to add a key word ephemeral that signal the ephemeral datatstore for items in the config true or the config false state.

```
   module thermostat {
     ..
     ! Do we need an ephemeral flag here for consistency (??sue)
     !
     leaf desired-temp {
        type int32;
            units "degrees Celsius";
            description "The desired temperature";
            ephemeral true;
            }

     leaf actual-temp {
        type int32;
            config false;
            ephemeral true;
            units "degrees Celsius";
            description "The measured temperature";
            }
     }
```

Figure 8 - Simple Thermostat Yang with ephemeral

Figure 6 shows the thermostat model has emphemeral variable desired-temp in the running configuration and the ephemeral data store. The RESTCONF way of addressings is below:

RESTCONF running data store

```
PUT /resconf/data/thermostat:desired-temp
{"desired-temp":18}
```

RESTCONF ephemeral datastore

```
PUT /restconf/data/thermostat:desired-temp?datastore=ephemeral
{"desired-temp":19 }
```

Figure 7 shows the thermostat model with an addition of the actual-temp in the ephemeral operational store that would be stored in the actual operational status. The RESTCONF syntax is below:

RESTCONF Ephemeral Datastore Edit of Config=FALSE

```
PUT /restconf/data/thermostat:actual-temp?datastore=ephemeral
{"actual-temp":72}
```

7. NETCONF protocol extensions for the ephemeral datastore

capability-name: ephemeral-datastore

7.1. Overview

This capability defines the NETCONF protocol extensions for the ephemeral state. The ephemeral state has the following features:

- the ephemeral datastore is a datastore holds configuration that is intended to not survive a reboot.

- The ephemeral datastore is never locked.

- Each client has a unique priority.

- Each client writes into its own pane so there is no conflict within a pane. The implementation combines the panes into the appropriate image.

- A Partial operation is one where a subset of the written data is not applied because of better priority for that node. A partial operation is only allowed if the error-option is stop-on-error or continue-on-error.

- Caching is optional and and a server may retain the pain for each client.

7.2. Dependencies

The Yang data modules must be flag with the ephemeral data store. The Yang modules must support the notification of write-conflicts.

7.3. Capability identifier

The ephemeral-datastore capability is identified by the following capability string: (capability uri)

7.4. New Operations

7.4.1. Bulk-write

The bulk-write goes here.

7.4.2. Bulk-Read

The bulk-read goes here.

7.5. Modification to existing operations

7.5.1. PUT changes

The phrase "?datastore=ephemeral" following an element will specify the ephemeral data store.

7.6. Interactions with Other Capabilities

TBD

8. RESTCONF protocol extensions for the ephemeral datastore

capability-name: ephemeral-datastore

8.1. Overview

This capability defines the REST CONF protocol extensions for the ephemeral state. The ephemeral state has the following features:

- the ephemeral datastore is a datastore holds configuration that is intended to not survive a reboot.

- The ephemeral datastore is never locked.

- Each client has a unique priority.

- Each client writes into its own pane so there is no conflict within a pane. The implementation combines the panes into the appropriate image.

- A Partial operation is one where a subset of the written data is not applied because of better priority for that node. A partial operation is only allowed if the error-option is stop-on-error or continue-on-error.

- Caching is optional and and a server may retain the pain for each client.

8.2. Dependencies

The Yang data modules must be flag with the ephemeral data store. The Yang modules must support the notification of write-conflicts.

8.3. Capability identifier

The ephemeral-datastore capability is identified by the following capability string: (capability uri)

8.4. New Operations

8.4.1. Bulk-write

The bulk-write goes here.

8.4.2. Bulk-Read

The bulk-read goes here.

8.5. Modification to existing operations

8.5.1. PUT changes

The phrase "?datastore=ephemeral" following an element will specify the ephemeral data store.

8.6. Interactions with Other Capabilities

TBD

9. IANA Considerations

TBD

10. Security Considerations

TBD

11. Acknowledgements

This document is an attempt to distill lengthy conversations on the I2RS proto design team from August

Here’s the list of the I2RS protocol design team members

- Alia Atlas
- Andy Bierman
- Alex Clemm
- Eric Voit

- Kent Watsen
- Jeff Haas
- Keyur Patel
- Hari
- Dean Bogdanavich
- Anu Nair
- Juergen Schoenwaelder
- Kent Watsen

12. References

12.1. Normative References:

[I-D.hares-i2rs-auth-trans]
Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-hares-i2rs-auth-trans-05 (work in progress), August 2015.

[I-D.ietf-i2rs-architecture]
Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-09 (work in progress), March 2015.

[I-D.ietf-i2rs-pub-sub-requirements]
Voit, E., Clemm, A., and A. Prieto, "Requirements for Subscription to YANG Datastores", draft-ietf-i2rs-pub-sub-requirements-02 (work in progress), March 2015.

[I-D.ietf-i2rs-rib-info-model]
Bahadur, N., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-07 (work in progress), September 2015.

[I-D.ietf-i2rs-traceability]
Clarke, J., Salgueiro, G., and C. Pignataro, "Interface to the Routing System (I2RS) Traceability: Framework and Information Model", draft-ietf-i2rs-traceability-03 (work in progress), May 2015.

[I-D.ietf-netmod-yang-metadata]
Lhotka, L., "Defining and Using Metadata with YANG", draft-ietf-netmod-yang-metadata-02 (work in progress), September 2015.

[I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

12.2. Informative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-07 (work in progress), July 2015.

[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>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[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, <http://www.rfc-editor.org/info/rfc6241>.

[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>.

Authors’ Addresses

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

Andy Bierman
Juniper

Kent Watsen
Juniper

Email: kwatsen@juniper.net