I2RS working group
Internet-Draft
Intended status: Informational
Expires: January 9, 2017

S. Hares
Huawei
A. Dass
Ericsson
July 8, 2016

I2RS protocol strawman
draft-hares-i2rs-protocol-strawman-03.txt

Abstract

This strawman proposal discusses requirement, design, and implementation issues for an I2RS protocol which supports I2RS requirements for ephemeral data store, management data flows, and protocol security. It proposes additions to the NETCONF, RESTCONF, and YANG for these requirements.

This document is a living document providing insights gained in design, implementation and debugging of the I2RS protocol.

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 January 9, 2017.

Table of Contents

1. Introduction 3
2. Definitions Related to Ephemeral Configuration 4
2.1. Requirements language 4
2.2. I2RS Definitions 5
2.3. Operational and Ephemeral State definitions 6
3. DataStore Model Melee 7
3.1. Opstate model 7
3.2. Persistent/Non-Persistent Config 10
3.3. Revised Data Store (Ephemeral is OPSTATE 11
3.4. Another Model for Ephemeral 13
4. Summary of Protocol Changes 14
4.1. Ephemeral Data 14
4.1.1. Overview of Ephemeral Data Store 14
4.1.2. I2RS Agent Caching of Ephemeral Data 15
4.2. Protocol Security 15
4.2.1. Summary of Protocol Security Changes 15
4.3. Data Flow 17
4.3.1. Data Flow for Ephemeral Configuration 17
4.3.2. Write Error handling 18
4.3.3. Data Flows From the I2RS Agent to I2RS Client 22
4.3.4. OAM Constraints 23
4.3.5. IPFIX for traffic monitoring 23
4.4. Yang Changes 23
4.5. Transport Protocol Changes 24
4.5.1. Secure Protocols 24
4.5.2. Insecure Protocol 24
4.6. NETCONF protocol extensions for the ephemeral datastore . 24
4.6.1. Overview 25
4.6.2. Dependencies 26
4.6.3. Capability identifier 26
4.6.4. New Operations 26
4.6.5. Modification to existing operations 26
4.6.6. Interactions with Capabilities 29
4.7. RESTCONF protocol extensions for the ephemeral datastore 31
4.7.1. Overview 31
4.7.2. Dependencies 31
4.7.3. Capability identifier 32
4.7.4. New Operations 32
4.7.5. modification to data resources 32
4.7.6. Modification to existing operations 32
4.7.7. Interactions with Notifications 33

4.7.8. Interactions with Error Reporting 33
5. Simple Thermostat Model 34
5.1. YANG data model 35
5.2. NETCONF Changes 40
5.3. RESTCONF Initial Write 40
6. Simple Route Add 40
6.1. Portions of I2RS YANG data model 43
6.2. NETCONF Changes 45
6.3. RESTCONF Changes 45
7. Previously Considered Ideas 45
7.1. A Separate Ephemeral Data store 45
7.2. Panes of Glass/Overlay 46
8. IANA Considerations 46
9. Security Considerations 46
10. Acknowledgements 46
11. Major Contributors 47
12. List of I2RS Requirements 47
13. References 47
13.1. Normative References: 48
13.2. Informative References 49
Authors' Addresses 51

1. Introduction

This is a strawman proposal for the first version of the I2RS protocol. This draft is input to a NETCONF Working Group which standardizes extensions to the NETCONF and RESTCONF protocol, and to the NETMOD Working Group which standardizes extensions to YANG. This draft is also input to the early implementers of the I2RS protocol. As such, feedback from early implementations would help.

The I2RS protocol is a higher level protocol comprised of a set of existing protocols which have been extended to work together to support a new interface to the routing system. The I2RS protocol is a "reuse" management protocol which creates new management protocols by reusing existing protocols and extending these protocols for new uses. The first version of the I2RS protocols is comprised of extensions of the NETCONF [RFC6241] and RESTCONF [I-D.ietf-netconf-restconf].

This strawman proposal supports I2RS requirements for ephemeral data store, management data flows, and protocol security. It proposes extensions to the following:

- YANG 1.1 [I-D.ietf-netmod-rfc6020bis],

- NETCONF [RFC6241],

- RESTCONF [I-D.ietf-netconf-restconf]
- Network Access Control Model [RFC6536]

This protocol strawman utilizes the following existing proposed features for NETCONF and RESTCONF

- Call Home [I-D.ietf-netconf-call-home],
- Server Configuratino Module [I-D.ietf-netconf-server-model],
- Module library [I-D.ietf-netconf-yang-library],
- Publication/Subscription via Push [I-D.ietf-netconf-yang-push],
- Patch [I-D.ietf-netconf-yang-patch],
- syslog yang module (both [RFC5424] and [I-D.ietf-netmod-syslog-model]

Section 2 provides definitions for terms in this document. Section 3 summarizes the changes to configuration data store, NETCONF, RESTCONF, and YANG. Section 4 details the changes to Yang. Section 5 summarizes the changes to transport support for RESTCONF and NETCONF. Section 6 details the changes to NETCONF. Section 7 details the changes to RESTCONF. Section 8 provides a simple example of I2RS protocol support for the ephemeral data store using a simple temperature model. Section 9 provides a simple example of the I2RS protocol with an ephemeral route updating an existing route. Section 10 provides information on the security considerations for the I2RS protocol.

2. Definitions Related to Ephemeral Configuration

This section reviews definitions from I2RS architecture [I-D.ietf-i2rs-architecture] and NETCONF operational state [I-D.ietf-netmod-opstate-reqs] before using these to construct a definition of the ephemeral data store.

2.1. 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].

2.2. I2RS Definitions

The I2RS architecture [I-D.ietf-i2rs-architecture] defines the following terms:

ephemeral data: is data which does not persist across a reboot (software or hardware) or a power on/off condition. Ephemeral data can be configured data or data recorded from operations of the router. Ephemeral configuration data also has the property that a system cannot roll back to a previous ephemeral configuration state.

local configuration: is the data on a routing system which does persist across a reboot (software or hardware) and a power on/off condition. Local configuration has the ability to roll back to a pervious configuration state.

operator-applied policy: is a policy that an operator sets that determines how ephemeral configuration interacts with local configuration. One could consider these policy knobs that the operator sets to determine how the I2RS agent will act. Two policy knobs are necessary:

* policy knob 1: Ephemeral configuration overwrites local configuration,

* policy knob 2: Updated configuration overwrites ephemeral configuration

Three possible setting for the above knobs are:

Policy knob 1=false and policy knob 2=true: I2RS software is installed, but the operator does not want it to overwrite write any configuration variables. This might be valid if I2RS is only suppose to monitor data on this node.

Policy knob 1=true and policy Knob 2=false: This is the normal case for the I2RS Agent where the ephemeral configuration data overwrites the local configuration data, and the ephemeral data stays even when the local configuration value changes. When the ephemeral data is removed by the I2RS agent, the most recent local configuration value is set.

Policy knob 1=true and Policy Knob 2=true: This case can occur if the ephemeral write is only suppose to take place until the next configuration cycle from a centralized system. Suppose the local configuration is get by the centralized system at 11:00pm each night. The I2RS Client writes temporary changes to the routing

system via the I2RS agent ephemeral write. At 11:00pm, the local configuration update overwrite the ephemeral. The I2RS Agent notifies the I2RS Client which is tracking which of the ephemeral changes are being overwritten.

2.3. Operational and Ephemeral State definitions

The [I-D.ietf-netmod-opstate-reqs] defines the following to augment [RFC6244] to define how configuration state and operational state are different.

Applied Configuration: This data represents the configuration state that the server is actually in.

Derived State: This data represents information which is generated as part of the server's own interactions.

Intended Configuration: This data is the configuration state that the network operator intends the server to be in, and that has been accepted by the server as valid configuration.

Operational State: is the current state of the system as known to the various components of the system (e.g., control plane daemons, operating system kernels, line cards). The operational state includes both applied configuration and derived state.

Ephemeral State: contains both ephemeral configuration state and operational state. In a data model which is only ephemeral, the operational data created based on ephemeral configuration state. In a non-ephemeral data model, the ephemeral augmentatation to operational state may create ephemeral operational state.

Ephemeral Configuration state: Ephemeral configuration state is state which is an ephemeral only data modules or ephemeral configuration that augments a non-ephemeral data model."

In each of these definitions, the "server" is the routing system.

The [I-D.ietf-netmod-opstate-reqs] defines two actions that update the intended and the applied configuration:

Asynchronous Configuration Operation: the server MUST update its intended configuration before replying to the client indicating whether the request will be processed. The server's applied configuration state is updated after the configuration change has been fully effected to all impacted components in the server.

Synchronous Configuration Operation: the server MUST fully attempt to apply the configuration change to all impacted components in the server, updating both its configuration (intended configuration and the applied configuration), before replying to the client.

3. DataStore Model Melee

The NETMOD Working Group has been working to create new definitions of datastores based on feedback from operators on desiring a split between operational state and configuration state.

In a system without ephemeral data, the structure of the routing systems local intended configuration, applied configuration, and derived state can be modeled in the following ways:

1. Opstate model from I-D.draft-kwatsen-netmod-opstate (figure 1 below),

2. Persistent/Non-Persistent Config from I-D.draft-wilton-netmod-refined-datastores (figure 2 below) ,

3. Revised Data Store (Ephemeral is OPSTATE) from I-D.draft-schoenw-netmod-revised-datastores (figure 3)

4. I2RS model (figure 4 below in section 2.4.1).

Section 2.4.1-2.4.4 provide diagrams and pro/cons of each model. Operational experience will improve our understanding of these datastore models.

3.1. Opstate model

Model from: I-D.draft-kwatsen-netmod-opstate

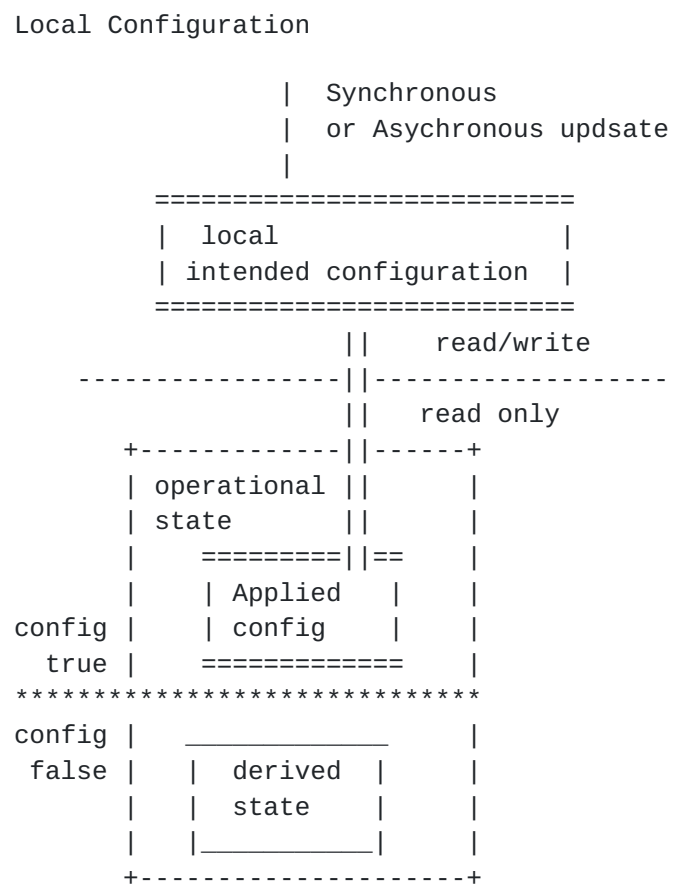


Figure 1

I2RS Proposed Ephemeral state addition to Model

Figure 2

Pro:

1. Ephemeral configuration utilize the constraint checks in yang 1.1 [I-D.ietf-netmod-rfc6020bis] in section 8.3 for message syntactial checks (8.3.1), and can use insertion and datastore based constraint checking in NETCONF/RESTCONF in their normal manner (section 8.3.2 and 8.3.3).

2. Operational state can occur in ephemeral only data models

3. Ephemeral augmentations seem a logical extension.

4. Event/notification for I2RS can work with config and ephemeral.

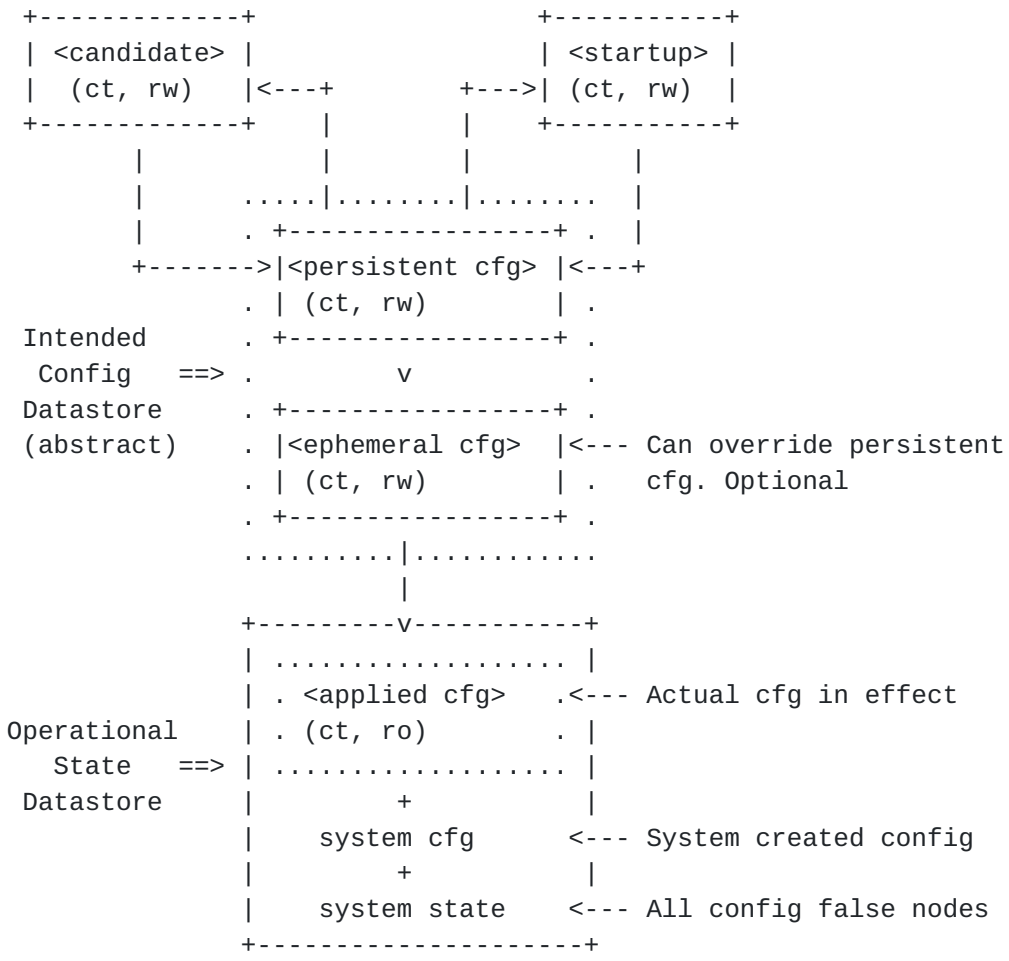
5. Can query ephemeral + local configurations as overlay or separately.

Con:

1. Ephemeral configuration cannot elect to use only the Yang 1.1 syntactical checks (section 8.3.1 of [I-D.ietf-netmod-rfc6020bis]).

3.2. Persistent/Non-Persistent Config

Persistent/Non-Persistent Config from I-D.draft-wilton-netmod-refined-datastores



Pro:

1. Ephemeral configuration utilize the constraint checks in yang 1.1 [I-D.ietf-netmod-rfc6020bis] in section 8.3 for message syntactial checks (8.3.1), and can use insertion and datastore based constraint checking in NETCONF/RESTCONF in their normal manner (section 8.3.2 and 8.3.3).

2. Operational state can occur in ephemeral only data models

3. Ephemeral augmentations align with the I2RS ephemeral requirements [I-D.ietf-i2rs-ephemeral-state]

4. Event/notification for I2RS can work with config and ephemeral.

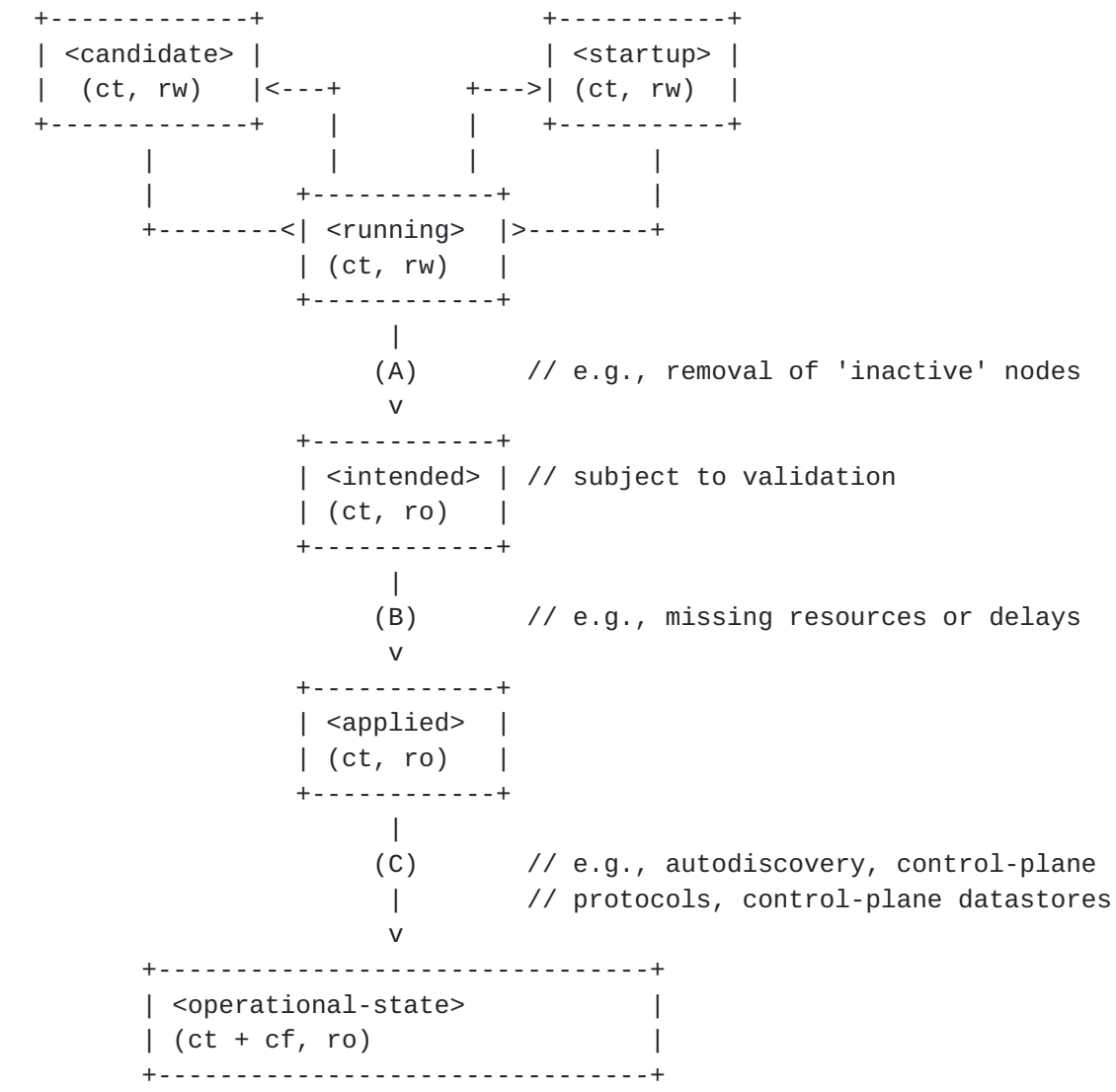
5. Can query ephemeral + local configurations as overlay or separately.

Con:

1. Ephemeral configuration cannot elect to use only the Yang 1.1 syntactical checks (section 8.3.1 of [I-D.ietf-netmod-rfc6020bis]).

3.3. Revised Data Store (Ephemeral is OPSTATE

Revised Data Store (Ephemeral is OPSTATE) from I-D.draft-schoenw-netmod-revised-datastores



Pro:

1. Ephemeral configuration can tailor its constraint checking to the data model.

2. Ephemeral data is like data sent to/from any routing process.

Con:

1. Ephemeral configuration state must create its own constraint checking

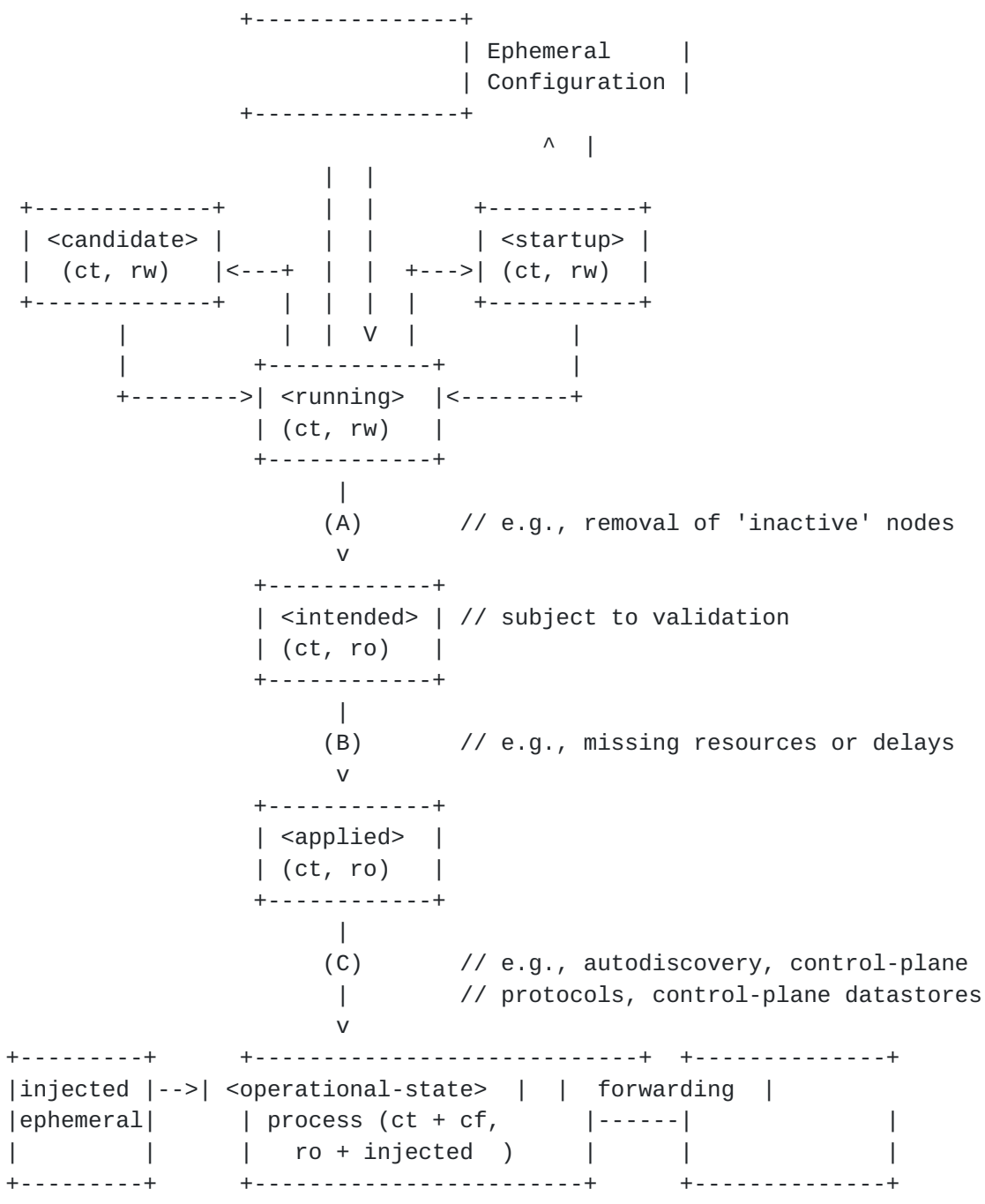
2. Ephemeral state has no easy way to query the overlay of ephemeral state and local configuration since there is no augmentation of local configuration.

3. Event/notification for I2RS can work with config and ephemeral.

4. Ephemeral configuration

3.4. Another Model for Ephemeral

This model came out of dicussions with the authors of all the drafts:



4. Summary of Protocol Changes

This section provides a summary of requirements for changes to support the I2RS protocol features of ephemeral data, a secure protocol, management data flows, and I2RS error handling. Management data flows may be large data flows for notifications, events, and protocol events. Management flows could also be tracing the routing system's operation or OAM operations.

4.1. Ephemeral Data

This section provides an overview of the ephemeral data store, I2RS agent caching support, and ephemeral requirements (from [I-D.ietf-i2rs-ephemeral-state]).

4.1.1. Overview of Ephemeral Data Store

This section augments the [I-D.ietf-netmod-opstate-reqs] with definitions for ephemeral state as the longest held NETMOD datastore model. Any of the ephemeral data models described above can be made to align to ephemeral state, but the revised data store (ephemeral is operation state) may take more effort.

NETCONF provides the concept of a data store, but RESTCONF only defines the concept of a "context". The logical description of ephemeral additions to the NETCONF data store below still fits the general concepts of the RESTCONF context.

This approach to the ephemeral datastore is two panes-of-glass model one pane of glass is the "local configuration" within the Intended configuration and the other pane of glass is the "ephemeral data". The two panes of glass are pressed together to create the intended configuration which then applied to the routing node and generates derived state as shown in figure 2.

The applied configuration is the result of the the intent configuration (normal and ephemeral). Similarly, the derived data is a result of the applied configuration (normal and ephemeral). Therefore derived state may be defined in local configuration or ephemeral portions of a data model (or data models).

The ephemeral data store has the following general qualities:

1. Ephemeral state is not unique to I2RS work.

2. The ephemeral datastore is never locked.

3. The ephemeral portion of the intended configuration, applied state, and derived state does not persist over a reboot,

4. an ephemeral node cannot roll-back to its previous value,

5. Since ephemeral data store is just data that does not presist over a reboot, then in theory any node or group of nodes in a YANG data model could be ephemeral. The YANG data module must indicate what portion of the data model (if any) is ephemeral.

   * A YANG data module could be all ephemeral (e.g. [I-D.ietf-i2rs-rib-data-model]) with no directly associated configuration models,

   * A YANG model could be all ephemeral but associated with a configuration model

   * or a single data node or data tree could be made ephemeral.

6. The management protocol (NETCONF/RESTCONF) needs to signal which poritons of a data model(node, tree, or data model) are ephemeral in the module library [I-D.ietf-netconf-yang-library].

4.1.2. I2RS Agent Caching of Ephemeral Data

I2RS protocol version does not support caching of ephemeral data the I2RS Agents. Future I2RS work MAY support caching of data in the I2RS Agents. Implementers are encourages to experiment with caching of ephemeral data in I2RS Agents.

4.2. Protocol Security

The I2RS protocol requires the ability to run over secure transport connections for the I2RS protocol to run over. Each secure transport must provide data confidentiality, data integrity, and replay prevention. NETCONF running over TLS or SSH over TCP, and RESTCONF running over HTTP 1.1 over TLS over TCP provide these features. However, the I2RS protocol requires extensions to this protocol security. This section provides an overview these changes.

4.2.1. Summary of Protocol Security Changes

The I2RS protocol requires the following new security features:

o mutual identification of I2RS Client and Agents via unique identifiers,

- the I2RS client identifier to be associated with a priority and a secondary identity
- data access (read/write) for each data model to be associated with I2RS client roles,
- the ability to send some data over an insecure section as specified in a data model.

This section describes these new features.

4.2.1.1. Multiple secure transports

The I2RS protocol MAY operate over a set of secure transports (1 to many transports) which provide data confidentiality, data integrity, and replay prevention. The key management that distributes keys MUST guarantee that only the entities having sufficient privileges can get the keys to encrypt/decrypt the sensitive data. NETCONF's operatoring over TLS or SSH protocols, both of which run over TCP, provide such a secure transport as does RESTCONF operating over HTTP 1.1 operating over TLS which runs over TCP also fits this description.

4.2.1.2. Mutual Identification

I2RS protocol security requires mutual identification of I2RS client and agent via a unique identifier. The identity of each I2RS client must be represented by at least one unique I2RS client identifier, and the identity of an I2RS Agent must be represented by at least one unique I2RS agent identifier. The I2RS protocol must perform mutual identification of the I2RS client and the I2RS agent. The I2RS client-agent security association is valid for a single transport session or a set of parallel transport sessions. The I2RS client-agent security association does not need to have an active transport session to remain active. The I2RS agent and client unique identifiers are created and distributed outside the I2RS protocol.

4.2.1.3. I2RS Client has Identifier + Priority + Secondary Identifier

Each I2RS client identifier will have one priority and one secondary identifier during a particular I2RS transaction (read/write sequence), but the priority and the secondary identity associated with a I2RS client identity may change during a I2RS client-agent association.

4.2.1.4. I2RS Role Based Access

Certain data within routing elements is sensitive and read/write operations on such data SHOULD be controlled as to which I2RS client can access the data for read/write based on the I2RS client's roles in order to protect its confidentiality. A I2RS Client's role describe which data models and which data within those data models the I2RS client can have read access, write access, or both (read/write).

4.2.1.5. Insecure Transport

An I2RS data model with ephemeral state MAY require the passage of I2RS data will require the some data to be be sent from the I2RS agent to a I2RS client via an insecure transport. Examples of this transport could be the I2RS agent agent opening up a TCP connection to an I2RS Client via TCP. The yang data model specifying this MUST indicate what data is able to be passed over an insecure transport connection. Insecure transport must still support traceability and publication/subscription of the insecure data.

4.3. Data Flow

Large amounts of data can flow from the I2RS agent to the I2RS client, or from the I2RS client to the I2RS Agent. The I2RS client may set or query ephemeral configuration in the routing system via the I2RS agent and receive operational state, notifications, or logging from the I2RS Agent on behalf of the I2RS routing system. In addition, some OAM functions engaged by the by the I2RS Client via the I2RS Agent can require large data flows plus system resources (cpu, memory, data storage). This section discusses implementation issues regarding this work.

4.3.1. Data Flow for Ephemeral Configuration

The requirements for high performance and high-volune data flow between the I2RS Clients and the I2RS agents in the routing system have been described in general in the I2RS architecture [I-D.ietf-i2rs-architecture]. lients can send large amount of ephemeral configuration data to the I2RS Agent. The writes may be done via NETCONF (<edit-config> or an rpc function), or via RESTCONF (PUT, PATCH, POST). Reads can be done via NETCONF <get-config> or RESTCONF GET or query.

The I2RS RIB Data Model [I-D.ietf-i2rs-rib-data-model] also supports the use of rpc to add/delete RIBs, add/delete/update routes, and add/delete nexthops. If the I2RS client does a small to medium number of writes to the I2RS ephemeral state in the I2RS Agent in a routing

system, the full validation that NETCONF or RESTCONF does will be able to be done without any reduction in speed to the I2RS high-performance system. For example, if the I2RS RIB Data Model has adds a 1000 routes, the I2RS RIB use of rpc to add/delete/update routes should be able to provide a high-performance system. Alternatively the NETCONF <edit-config> could update these 1000 routes with a write, or the RESTCONF POST, PUT or PATCH should be able to add the 1000 routes.

If a large number of ephemeral routes or filters are written (updates or new) by the I2RS Client to the ephemeral state in the I2RS agent, one of the key issues for a high performance interface is the time it takes to validate routes. Due to this concern, the I2RS architecture was design to allow less than the full NETCONF or RESTCONF validation. The concept is that the I2RS routes would be validated within the I2RS client and sent via a 99.999% reliable connection. In this scenario, the I2RS Agent would trust the validation that the I2RS Client did, and the communication of the route additions via the network connection.

An experiment regarding this has been done with the ODL code base update of ephemeral routes, but additional experimentation needs to be done prior to finalizing this design. Section 3.4.2 reviews how this process might be done, but many open issues exist in implementing this "low-validation" interface. Without additional experimentation and prototype code, this type of "low-validation",

4.3.2. Write Error handling

This section reviews I2RS normal error handling and error handling for rpc with no validation checks.

4.3.2.1. Normal validation checks

An I2RS agent validates an I2RS client's information by examining the following:

- message syntax validation,
- syntax validation for nodes of data model,
- referential checks (leafref checks MUST clauses, and instance indentifier),
- checks groups of data within a data model or groups of data across data models,
- write access to data,

- o if write access and values already exist, if I2RS client write access is higher than existing priority.

4.3.2.1.1. Multiple I2RS Clients Write Same Node

Multiple I2RS clients writing to the same variable is considered an "error condition" in the I2RS architecture [I-D.ietf-i2rs-architecture], but an I2RS Agent must handle this error condition. Upon multiple I2RS clients writing, the ephemeral data store allows for priority pre-emption of the write operation. Priority pre-emption means each I2RS client of the ephemeral I2RS agent (netconf server) is associated with a priority. Priority pre-emption occurs when a I2RS client with a higher priority writes a node which has been written by an I2RS client (with the lower priority). At this point, the I2RS agent (netconf server) allows the write and provides a notification indication to the notification publication/subscription service.

The I2RS protocol security requires that each I2RS client has a identity that has a unique identifier which has one priority and one secondary identitifer associated it during a write sequence (singel write or multiple group actions (see below).

An I2RS client's unique identifier is distributed along with valid roles and a valid priority via exterior mechanisms (AAA, administrative interface) to the I2RS agent. The secondary identifier is passed as an opaque meta value in the I2RS Client write. The exterior mechanism may change the the valid roles and priority associated with an I2RS client's identifier. If a change occurs after the I2RS client data has written information, the I2RS agent must revaluate the writes associate with this I2RS client (including rpcs). The I2RS agent may schedule this evaluation, but it should provide the following notifications to the I2RS client:

I2RS agent had received change of priority for I2RS client,

I2RS agent is beginning reevaluation of writes or rpcs associated with the client due to priority change,

I2RS agent has completed the revaluation due to priority change.

4.3.2.1.2. Multiple Action Messages

An I2RS agent receiving multiple action to write data within a message from an I2RS client must validate the data and check to make sure this I2RS client has permission and priority to change all the values. If one of the values in the multiple action messages fails

one of these tests, then error handling must decide what to do with the rest of the values.

Error handling in I2RS protocol version 1 simply remove all changed nodes and restores the previous values (all-or-nothing). In this case, the short term ephemeral values are kept until the message is processed.

Error handling on writes of the ephemeral datastore could be different for nodes that are grouped versus orthogonal. Group nodes may need to be all changed or all removed (all-or-nothing). In contrast, writing orthogonal data nodes in the same data module or between data models need to be added or deleted in sync, but the writes do not have to be "all-or-nothing."

I2RS suggests the following are some of the potential error handling techniques for multiple message sent to the I2RS client:

1. Perform all or none: All operations succeed or none of them will be applied. This useful when there are mutual dependencies.

2. Perform until error: Operations are applied in order, and when error occurs the processing stops. This is useful when dependencies exist between multiple-message operations, and order is important.

3. Perform all storing errors: Perform all actions storing error indications for errors. This method can be used when there are no dependencies between operations, and the client wants to sort it out.

The initial version of I2SR protocol restricts I2RS protocol implementations to an "all or nothing" (aka roll-back on error).

Is important to reliability of the data store that none of these error handling for multiple operations in one more multiple messages cause errors into be insert the I2RS ephemeral data-store.

Discussion of Current NETCONF/RESTCONF versus

RESTCONF does an atomic action within a http session, and NETCONF has atomic actions within a commit. These features may be used to perform these features.

I2RS processing is dependent on the I2RS model. The I2RS model must consider the dependencies within multiple operations work within a model.

4.3.2.1.2.1. Grouping and Error handling

Yang 1.1 provide the ability to group data in groupings, leafref lists, lists, and containers. Grouping of data within a model links to data that is logically associated with one another. Data models may logical group data across models. One example of such an association is the association of a static route with an interface. The concepts of groupings apply to both ephemeral and non-ephemeral nodes within a data model.

4.3.2.1.2.2. Why All-or-Nothing

NETCONF does not support a mandated sequencing of edit functions or write functions. Without this mandated sequences, NETCONF cannot support partial edits.

RESTCONF has a complete set of operations per message. The RESTCONF patch [I-D.ietf-netconf-yang-patch] could support partial edit functions per messages.

Since version 1 of I2RS protocol desires to support NETCONF and RESTCONF equally, the partial

4.3.2.1.2.3. Future Error Handling of Multiple Write Messages

The [I-D.ietf-i2rs-architecture] specifies three types of error handling for a partial write operation of orthogonal data:

- 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. (Inherent in all-or-nothing is the concept of checking all changes before applying.)

Grouped data must only use "all-or-nothing."

Future I2RS protocol versions will mandate "stop-on-error" handling or "continue-on-error" handling of multiple orthongal actions if a RESTCONF "patch" like facility is defined for NETCONF.

4.3.2.2. Reduced Validation (Experimental)

Note: It is not clear how reduced validation works handles all failures cases, so implementers are encouraged to experiment and report results. The authors are concerned about the various failures scenarios. One authors (Andy Bierman) raises concerns whether this can work with local configuration.

As described above, in some circumstances the I2RS client-agent communication may be considered almost perfect (99.999%), and the speed of update critical. In such cases, the operator may choose to have the I2RS client do all the validation within a group and between groups prior to downloading the data, and the I2RS agent to simply upload the data. One mechanisms puts this function in an RPCs

4.3.2.2.1. Reduced Validation RPC (Experimental)

The "no validation" feature requires:

- operator-applied policy knob enabling this feature;
- rpc in a data model with the yang "ephemeral-validation no-check;"

4.3.3. Data Flows From the I2RS Agent to I2RS Client

Large data flows can be required by the I2RS agent to publish large data for protocol state, virtual topologies, events, and notifications from a routing system. [I-D.ietf-i2rs-pub-sub-requirements] specify the I2RS requirements for publication of large data flows from the I2RS Agent via a publication/subscription (aka pub-sub) mechanism. The pub-sub mechanisms has been specified for the "push" service in [I-D.ietf-netconf-yang-push].

Large data flows can also be required to trace the actions of a routing system. These requirements are listed in the [I-D.ietf-i2rs-traceability]. These traceability requirements specify mandatory fields in the trace log including an end of message marker for a record plus handling of the trace logs. This handling includes creation of trace logs, limits on trace logs, trace log rotation, and trace log retrieval by syslog [RFC5424], the pub-sub mechanism or a large data push. This large data push can be a pull in a large write.

Large data flows from the I2RS client also mean that some of the data flows from the I2RS Agent may be prioritized over other data flows (I2RS-DF-REQ-07). This priorization will be based on what the data

is, what the operator-applied policy knobs are for reporting, and the current resource constraints (I2RS-DF-REQ-05).

4.3.4. OAM Constraints

OAM actions in a router may require extra processing, extra memory or data storage, or extra data flows to/from the I2RS agent. The OAM functions SHOULD not impact the routing functions so it cannot perform its main task of guiding the traffic. OAM functions must be able to be limited in terms of processing power, memory, data storage, or data flows to/from network (I2RS-DF-REQ-05).

4.3.5. IPFIX for traffic monitoring

Due to the potentially large data flow the traffic measurment statistics generate, these statistics are best handled by publication techniques within NETCONF or a separate protocol such as IPFIX. In the future version of the I2RS protocol may desire to support a data stream outbound from the I2RS Agent to an I2RS client via the IPFIX protocol.

4.4. Yang Changes

The data modules supporting the ephemeral datastore can use the Yang module library to describe their datastore. Figure 5 shows the module library data structure as found [I-D.ietf-netconf-yang-library].

The Proposed changes to Yang for I2RS protocol version 1 are:

- i2rs:version 1;
- i2rs:transport-nonsecure ok;
- i2rs:ephemeral-validation nocheck;
- ephemeral true;
- encoding [XML | JSON]
- protocol [RESTCONF | NETCONF]
- protocol-transport [ssh, tls, tcp]
- transport-ports [ports]

Since ephemeral data store, encoding methods, protocols, protocol transport, and transport ports are features of the general protocols, these are not tagged with the "i2rs:" key word.

4.5. Transport Protocol Changes

4.5.1. Secure Protocols

NETCONF's XML-based protocol ([RFC6241]) can operate over the following secure and encrypted transport layer protocols:

SSH as defined in [RFC6242],

TLS with X.509 authentication [RFC7589]

RESTCONF's XML-based or JSON [RFC7158] data encodings of Yang functions are passed over HTTOS with (GET, POST, PUT, PATCH, DELETE, OPTIONS, and HEAD).

4.5.2. Insecure Protocol

The ephemeral database may support insecure protocols for information which is ephemeral state which does not engage in configuration. The insecure protocol must be defined in conjunction with a data model or a subdata model.

[RFC6536] with extensions supporting ephemeral, non-secure transport, and rpcs with no validation checks might look like:

```
extension ephemeral {
 description "if present in a data definition statement
    then the object is considered OK for editing as ephemeral data."
        }
extension non-secure-ok {
  description "if present in data definition statement
   then the object is considered OK for non-secure transport."
   }
extension ephemeral-validation-nocheck {
  description "if present in rpc definition
  the data received in the rpc is considered to
  not require validation checks.
   }
```

4.6. NETCONF protocol extensions for the ephemeral datastore

capability-name: ephemeral-datastore

4.6.1. Overview

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

- the ephemeral data store is a part of the intended configuration datastore, applied configuration datastore, and the derived state store whose components are not survive a reboot.

- The ephemeral capability is signalled as a capability of a leaf, grouping, a sub-module, or module that is stored as a feature of the module in the netconf yang module library ([I-D.ietf-netconf-yang-library]) used by Yang 1.1 and RESTCONF and NETCONF.

- ephemeral data will be noted by an "ephemeral" statement in for a leaf, grouping, sub-module, or module.

- The ephemeral datastore is never locked.

- The ephemeral data store is one pane of glass that overrides the local configuration (which is considered one pane of glass) in the intended config based on operator-applied policy knobs (see section 2.1).

- Ephemeral data can occur as part of protocol or protocol independent modules. However, ephemeral data nodes cannot have non-ephemeral data nodes within the subtree. Ephemeral sub-modules cannot have non-ephemeral data nodes within the module. Ephemeral modules cannot have non-ephemeral sub-modules or nodes within the module. Yang 1.1 [I-D.ietf-netmod-rfc6020bis] augmented by ephemeral state must enforce this restriction. Similarly, the Yang mount schema [I-D.ietf-netmod-schema-mount] must check for this restriction.

- Ephemeral writes should enforce the normal validation checks, priority pre-emption error handling if multiple I2RS clients write the same data, and "all-or-nothing" error handling for multiple actions in a write for data in groupings or orthogonal data (see section 3.4). The I2RS agent should send the I2RS client requesting write the notification of any type of error during the write process: failure of normal validation, priority pre-emption causing failure to write, multiple actions causing failure to sustain write (aka all-or-nothing roll-back). If the I2RS agent allows a priority pre-emption of the write of data model value by an I2RS client (e.g. client 1) of another I2RS client (e.g. client 2), then the I2RS agent must send a notification of the I2RS pre-emption to the previous I2RS client (e.g. client 2).

- Ephemeral writes as part of an rpc should allow the rpc to skip normal validation checks if data model specifies "ephemeral-validation nocheck;". The rpc which skips the normal validation MUST resolve the pre-emption write error handling for any data being written without normal validation check, and MUST only all the data within a grouping rather than orthogonal data.

4.6.2. Dependencies

The following are the dependencies for ephemeral support:

- The Yang definitions specified in section 6.
- The Yang modules must support the event notification write and read errors as well as data model errors.
- The following features must be supported by NETCONF
  - Call Home [I-D.ietf-netconf-call-home],
  - Server Configuratino Module [I-D.ietf-netconf-server-model],
  - Module library [I-D.ietf-netconf-yang-library],
  - Publication/Subscription via Push [I-D.ietf-netconf-yang-push],
  - Patch [I-D.ietf-netconf-yang-patch],
  - syslog yang module (both [RFC5424] and [I-D.ietf-netmod-syslog-model]

4.6.3. Capability identifier

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

4.6.4. New Operations

None

4.6.5. Modification to existing operations

The capability for :ephemeral-datastore modifies the target for existing operations.

4.6.5.1. <get-config>

The :ephemeral-datastore capability modifies the <edit-config> to accept the <ephemeral> as a target for source, and allows the filters focused on a particular module, submodule, or node.

The positive and negative responses remain the same.

Example - retrieve users subtree from
ephemeral database

```
 <rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-config>
      <source>
        <emphemeral-datastore/>
      </source>
      <filter type="subtree">
        <top xmlns="http://example.com/schema/1.0/thermostat/config">
        <desired-temp>
         </top>
      </filter>
   </get-config>
 </rpc>
```

4.6.5.2. <edit-config>

The :ephemeral-datastore capability modifies the <edit-config> to accept the <ephemeral> as a target for source with filters. The operations of merge, replace, create, delete, and remove are available, but each of these operations is modified by the priority write as follows:

<merge> parameter is replaced by <merge-priority> The current data is modified by the new data in a merge fashion only if existing data either does not exist, or is owned by a lower priority client. If any data is replaced, this event is passed to the notification function within the pub/sub and traceability.

<replace> is replaced by <replace-priority> for ephemeral datastore which replaces data if the existing data is owned by a lower priority client. If data any data is replaced, this event is passed to the notification function within pub/sub and traceability for notification to the previous client. The success or failure of the event is passed to traceabilty.

<create> - the creation of the data node works as in [RFC6241] except that the success or failure is passed to pub/sub and traceability functions.

<deletion> - the deletion of the data node works as in [RFC6241] except event that the success or the error event is passed to the notiication services in the pub/sub and traceability functions.

<remove> - the remove of the data node works as in [RFC6241] except that all results are forwarded to traceabilty.

The existing parameters are modified as follows:

<target> - add a target of :emphemeral-datastore

<default-operation> -allows only <merge-priority> or <replace-priority>

<error-option> - the I2RS agent agent supports only the a"all-or-nothing" equivalent to a "rollback-on-error" function.

positive response - the <ok> is sent for a positive response within an <rpc-reply>.

negative response - the <rpc-error> is sent for a negative response within an <rpc-reply>. Note a negative respones may evoke a publication of an event.

4.6.5.3. <copy-config>

Copy config allows for the complete replacement of all the ephemeral nodes within a target. The alternation is that source is the :ephemeral datastore with the filtering to match the datastore. The following existing parameters are modified as follows:

<target> - add a target of :emphemeral-datastore

<error-option> - the I2RS agent agent supports only the a"all-or-nothing" equivalent to a "rollback-on-error" function.

positive response - the <ok> is sent for a positive response within an <rpc-reply>.

negative response - the <rpc-error> is sent for a negative response within an <rpc-reply>.

4.6.5.4. <delete-config>

The delete will delete all ephemeral nodes out of a datastore. The target parameter must be changed to allow :ephemeral-datastore. and filters.

4.6.5.5. <lock> and <unlock>

Lock and unlock are not supported with a target of :ephemeral-datastore.

4.6.5.6. <get>

The <get> is altered to allow a target of :ephemeral-datastore and with the filters.

4.6.5.7. <close-session> and <kill-session>

The close session is modified to take a target of :ephemeral-datastore, Since no locks are set, none should be released.

The kill session is modified to take a target of "ephemeral-datastore. Since no locks are set, none should be released.

4.6.6. Interactions with Capabilities

[RFC6241] defines NETCONF capabilities for writeable-running datastore, candidate config data store, confirmed commit, rollback-on-error, validate, distinct start-up, URL capability, and XPATH capability. I2RS ephemeral state does not impact the writeable-running data store or the candiate config datastore.

4.6.6.1. writable-running and candidate datastore

The writeable-running and the candidate datastore cannot be used in conjunction with the ephemeral data store. Ephemeral database overlays an intended configuration, and does not impact the writable-running or candidate data store.

4.6.6.2. confirmed commmit

Confirmed commit capability is not supported for the ephemeral datastore.

4.6.6.3. rollback-on-error

The rollback-on-error when included with ephemeral state allows the error handling to be "all-or-nothing" (roll-back-on-error).

4.6.6.4. validate

The validation function operates normally with one addition with one addition for any data handled by an rpc with "ephemeral-validation nocheck".

The rpc specifying ephemeral-validation nocheck MUST specify within the ephemeral data written by the rpc function the following grouping:

```
  grouping ephemeral-validation-notcheck {
        leaf rpc {
          type string rpc-id;
          description "rpc wrote
           the non-check data";
        }
    leaf rpc-seq {
           type uint32 rpc-id;
           description "sequence number of
            rpc that wrote non-check data";
        }
    leaf client-id {
      type uint64 client-id;
          description "client identifier
           that wrote non-checking rpc;"
        }
    description "Tracking on rpc with
      no validation checking so validation
      failure can send note to client.";
  };
```

If the data validation finds an error in a component that was non-check, the notification should include the data module, submodule (if valid).

(Editor's note: Initial experiments on this type of rpc for I2RS RIB routes and I2RS FB-RIB filters will be done before IETF 96.

4.6.6.5. Distinct Startup Capability

This NETCONF capability appears to operate to load write-able running config, running-config, or candidate datastore. The ephemeral state does not change the environment based on this command.

4.6.6.6. URL capability and XPATH capability

The URL capabilities specify a <url> in the <source> and <target>. The initial suggestion to allow both of these features to work with ephemeral datastore.

4.7. RESTCONF protocol extensions for the ephemeral datastore

capability-name: ephemeral-datastore

4.7.1. Overview

This capability defines the RESTCONF protocol extensions for the ephemeral state. The ephemeral state has the features described in the previous section on NETCONF.

4.7.2. Dependencies

The ephemeral capabilities have the following dependencies:

- The Yang definitions specified in section 6.
- The Yang modules must support the event notification write and read errors as well as data model errors.
- The following features must be supported by RESTCONF
  - Call Home [I-D.ietf-netconf-call-home],
  - Server Configuratino Module [I-D.ietf-netconf-server-model],
  - Module library [I-D.ietf-netconf-yang-library],
  - Publication/Subscription via Push [I-D.ietf-netconf-yang-push],
  - Patch [I-D.ietf-netconf-yang-patch],
  - syslog yang module (both [RFC5424] and [I-D.ietf-netmod-syslog-model]

4.7.3. Capability identifier

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

4.7.4. New Operations

none

4.7.5. modification to data resources

RESTCONF must be able to support the ephemeral datstore as a context with its rules as part of the "{+restconf}/data" subtree. The "edit collision" features in RESTCONF must be able to provide notification to I2RS read functions or to rpc functions. The "timestamp" with a last modified features must support the traceability function.

The "Entity Tag" could support saving a client-priority tuple as a opaque string, but it is important that that additions be made to restore client-priority so it can be compared with strimgs can be done to determine the comparison of two I2RS client-priorities.

4.7.6. Modification to existing operations

The current operations in RESTCONF are: OPTIONS, HEAD, GET, POST, PUT, PATCH, and DELETE. This section describes the modification to these exiting operations.

4.7.6.1. OPTIONS changes

The options methods should be augmented by the [I-D.ietf-netconf-yang-library] information that will provide an indication of what ephemeral state exists in a data modules, or a data modules sub-modules or nodes.

4.7.6.2. HEAD changes

The HEAD in retrieving the headers of a resources. It would be useful to changes these headers to indicate the datastore a node or submodule or module is in (ephemeral or normal), and allow filtering on ephemeral nodes or trees, submodules or module.

4.7.6.3. GET changes

GET must be able to read from the URL and a context ("?context=ephemeral"). Similarly, it is important the Get be able to determine if the context=ephemeral.

4.7.6.4. POST changes

POST must simply be able to create resources in ephemeral datastores ("context=ephemeral") and invoke operations defined in ephemeral data models.

4.7.6.5. PUT changes

PUT must be able to reference an ephemeral module, sub-module, and nodes ("?context=ephemeral").

4.7.6.6. PATCH changes

Plain PATCH must be able to update or create child resources in an ephemeral context ("?context=ephemeral") The PATCH for the ephemeral state must be change to provide a merge or update of the original data only if the client's using the patch has a higher priority than an existing datastore's client, or if PATCH requests to create a new node, sub-module or module in the datastore.

4.7.6.7. DELETE changes

The phrase "?context=ephemeral" following an element will specify the ephemeral data store when deleting an entry.

4.7.6.8. Query Parameters

The query parameters (content, depth, fields, insert, point, start-time, stop-time, and with-defaults (report-all, trim, explicit, report-all-tagged) must support ephemeral context ("?context=ephemeral") described above.

4.7.7. Interactions with Notifications

The ephemeral database must support the ability to publish notifications as events and the I2RS clients being able to receiving notifications as Event stream. The event error stream processing should support the publication/subscription mechanisms for ephemeral state defined in [I-D.ietf-netconf-yang-push].

4.7.8. Interactions with Error Reporting

The ephemeral database must support in RESTCONF must also support passing error information regarding ephemeral data access over to RESTCONF equivalent of the and traceability client.

5. Simple Thermostat Model

In this discussion of ephemeral configuration, this draft utilizes a simple thermostat model with the YANG configuration found in figure 6. The desired-temp is local configuration node that has an ephemeral The actual temperature is a derived state node that records the actual temperature of the room.

Figure 6 shows two I2RS clients. I2RS client 1 has one connection to write the ephemeral copy of the desired temperature at priority 1. I2RS client 2 writes to the intended configuration with priority 10. I2RS client 1 has a second connetion to read the actual temperature, and I2RS client 2 also has a second connection to read the actual temperature.

The NETCONF example shows a simple write of the ephemeral state value over the local configuration

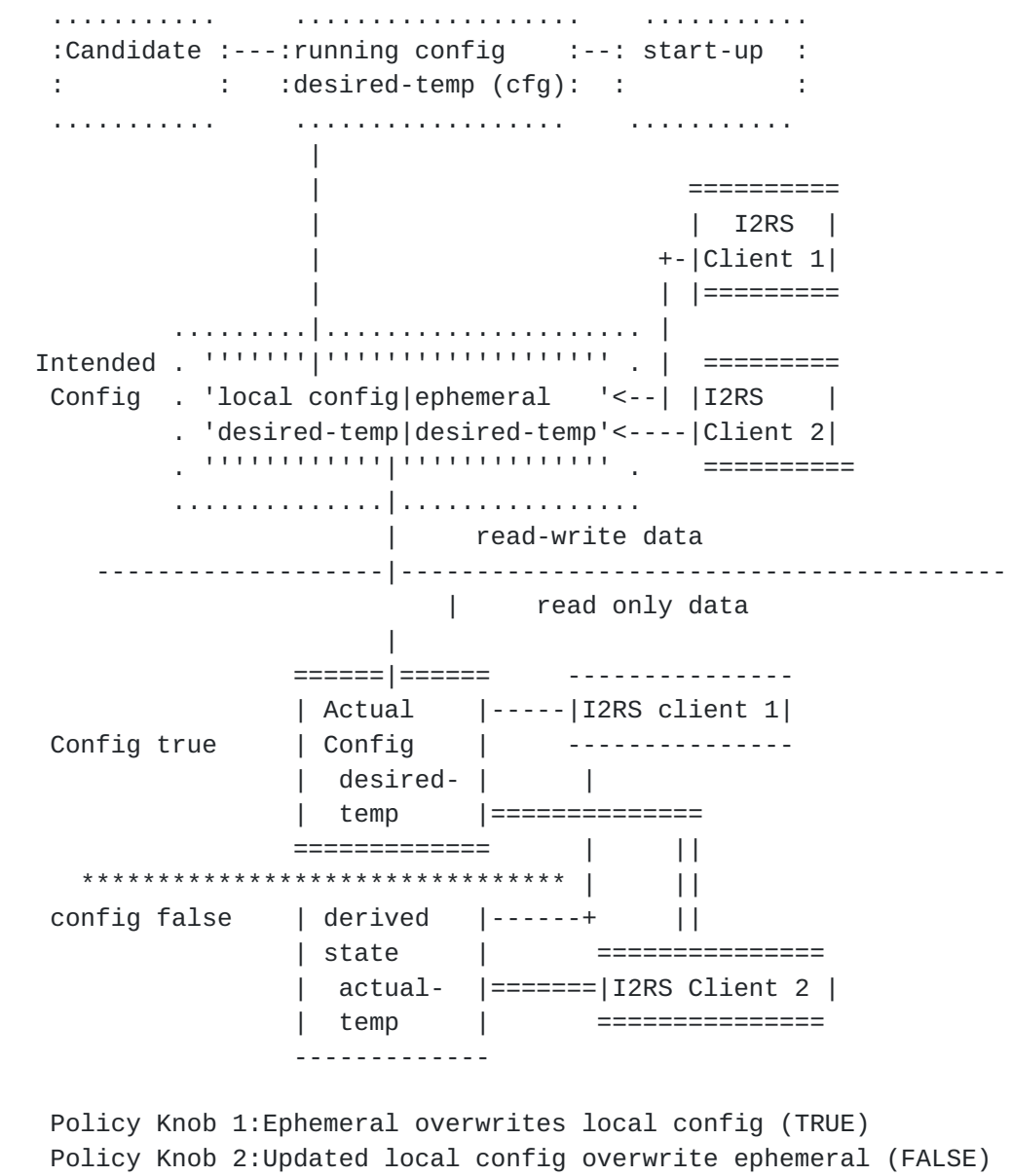


Figure 6 - Two I2RS clients

5.1. YANG data model

```
module thermostat {
  ..
  leaf desired-temp {
     type int32;
         config true;
         ephemeral true;
         units "degrees Celsius";
         description "The desired temperature";
         }
  ....
  leaf actual-temp {
     type int32;
         config false;
         units "degrees Celsius";
         description "The measured
         temperature is derived state.
         }
  }
```

Figure 6 - Simple thermostat YANG Model

The changes in each step are shown in the figure 7. In step 1, the running configuration desired-temp is change to 68 degress. In step 2, the intended configuration value for desired-temp is updated, and asynchronously the applied configuration is updated in step 4. The actual temperature begins to rise to meet the desired temperature, and reaches it in step 4. In step 5, I2RS client 1 update the intended configuration with a desired-temp=70. In step 6 this value is updated to the applied configuration, and the actual temperature begins to rise (actual-temp = 69). In step 7, the actual temperature has reached 70 degrees. In step 8, I2RS Client 1 removes the ephemeral state from the intended configuration and the local configuration value is reasserted. In step 9, the intended desired-temp is synchronously moved to applied configuration and the actual temperature drops.

Step	Running	Intended Config	Applied Config	Derived state
1	desired-temp=68			actual-temp=65
2	desired-temp=68	from running desired-temp temp = 68		actual-temp=65
3	Desired temp=68	Desired temp=68	Desired temp=68	Actual-temp=67
4	Desired temp=68	Desired temp=68	Desired temp=68	Actual-temp=68
5	Desired temp=68	from I2RS client 1 Desired temp=70	Desired temp=68	Actual-temp=68
6	Desired temp=68	Desired temp=70	Desired temp=70	Actual-temp=69
7	Desired temp=68	Desired temp=70	Desired temp=70	Actual-temp=70
8	Desired temp=68	I2RS client 1 removes state Desired temp=68	Desired temp=70	Actual-temp=70
9	Desired temp=68	Desired temp=68	Desired temp=68	Actual-temp=68

Figure 7

I2RS Client 1 handle the normal lowering and raising of the temperature during different time periods in the day. I2RS Client 2 has the ability for individuals to request the room warms up rapidly to a maximum of 72 degrees. Figure 8 shows a simple example of the two clients interaction. Steps 1-6 are the same as in figure 7. In step 7, I2RS Client 2 sets the desired-temp in the intended configuration to 72. In step 8, this intended configuration is passed to the applied configuration and the actual temperature reaches 72.

In step 9, I2RS client 2 removes its state. The I2RS Client 1 is notified of the removal, and the I2RS Client 1 re-write the desired value of 70 degrees (desired-temp=70), and this is passed to the applied state. The actual temperature drops to 70 degress (actual-temp=70). In step 10, I2RS Client 1 removes its ephemeral state and desired-temp reverts to the local configuration value (desired=temp=68). This value is installed in applied temperature and the actual temperature goes to 68 (actual-temp=68.)

Step	Running	Intended Config	Applied Config	Derived state
1	desired-temp=68			actual-temp=65
2	Desired temp=68	from running desired-temp temp = 68		actual-temp=65
3	Desired temp=68	Desired temp=68	Desired temp=68	Actual-temp=67
4	Desired temp=68	Desired temp=68	Desired temp=68	Actual-temp=68
5	Desired temp=68	from I2rs client 1 Desired temp=70	Desired temp=68	Actual-temp=68
6	Desired temp=68	Desired temp=70	Desired temp=70	Actual-temp=69
7	Desired temp=68	I2RS Client 2 sets Desired temp=72	Desired temp=70	Actual-temp=70
8	Desired temp=68	Desired temp=72	Desired temp=72	Actual-temp=72
		I2RS client 2 removes state reverts to I2RS client 1		
9	Desired temp=68	Desired temp=70	Desired temp=70	Actual-temp=70
		I2RS client 1 removes state		
10	Desired temp=68	Desired temp=68	Desired temp=68	Actual-temp=68

Figure 8

5.2. NETCONF Changes

The NETCONF way of writing the ephemeral data to the intended configuratino would be

```
<rpc-message-id=101
  xmlns="urn:ietf:params:xml:ns:base:1.0">
  <edit-config>
    <target>
     <ephemeral >
            true
         </ephemeral >
    </target>
    <config>
      <top xmlsns="http:://example.com/schema/1.0/thermostat/config>
       <desired-temp> 70 </desired-temp>
      </top>
    </config>
   </edit-config>
</rpc>
```

figure 9 NETCONF setting of desired-temp

5.3. RESTCONF Initial Write

Figure 10 shows the thermostat model has ephemeral variable desired-temp in the running configuration and the ephemeral data store. The RESTCONF way of addressing is below:

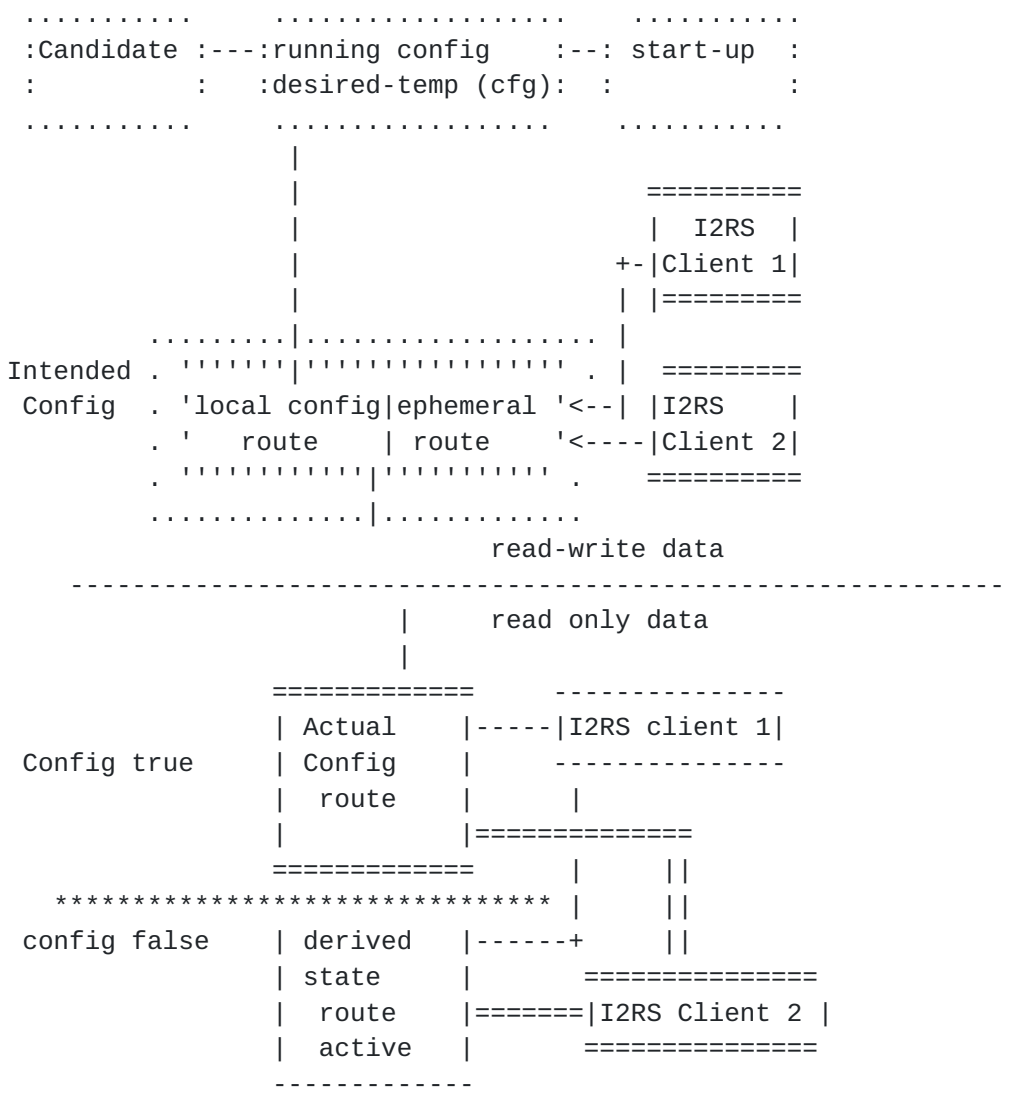
```
 RESTCONF ephemeral datastore

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

Figure 8 - RESTCONF setting of ephemeral state

6. Simple Route Add

In this discussion of ephemeral configuration, this draft utilizes the I2RS RIB data model [I-D.ietf-i2rs-rib-data-model] where one client adds an route via a rpc to the I2RS ephemeral data model.

Figure 9 shows two I2RS clients. I2RS client 1 writes ephemeral routes with priority 1, and I2RS client 2 writes ephemeral routes with priority 5. I2RS Client 1 and I2RS client can read the I2RS RIB With its status of installation. For ease of display the I2RS client 1 is show as two separate boxes, but these boxes are logically one client. Client 2 is also shown as two boxes, but has only one box.



Policy Knob 1:Ephemeral overwrites local config (TRUE)
Policy Knob 2:Updated local config overwrite ephemeral (FALSE)

Figure 11 - Two I2RS clients

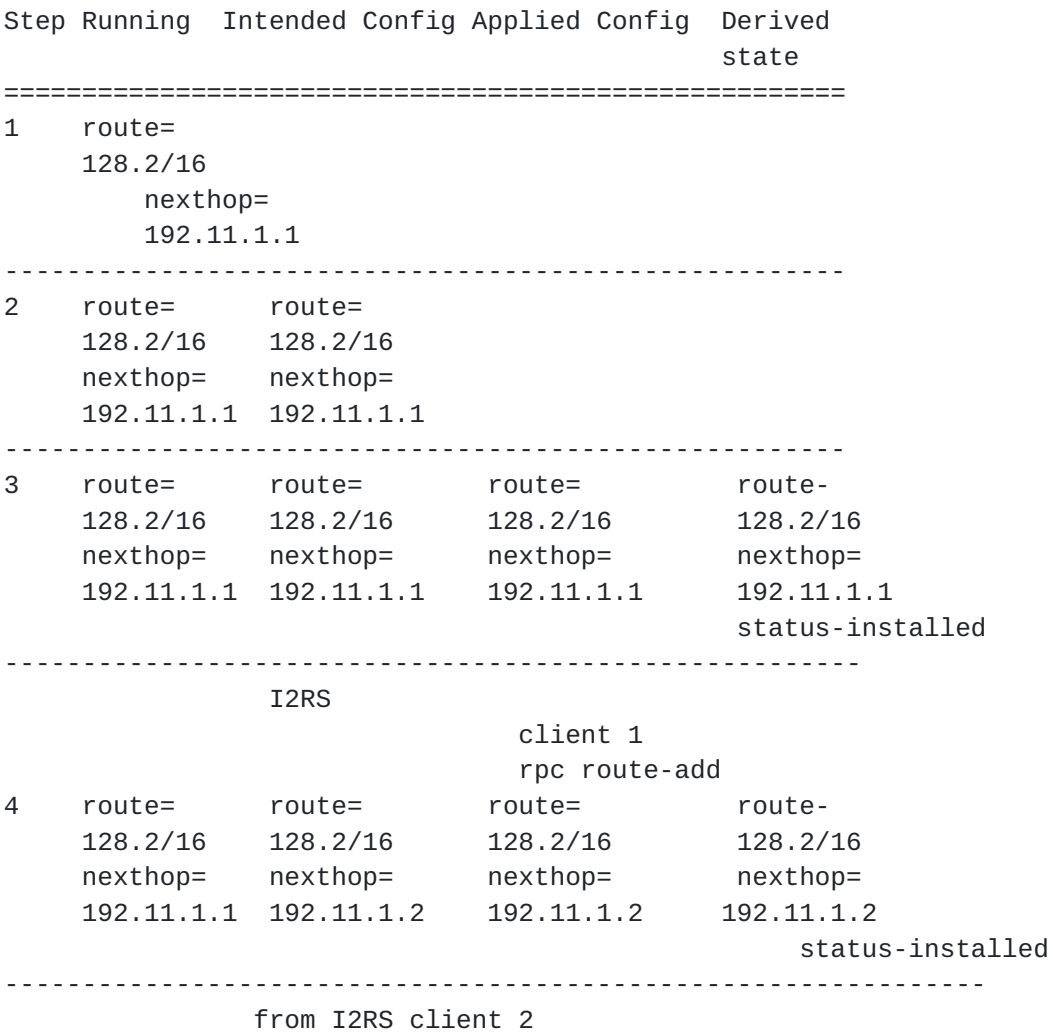
Figure 10 shows the addition of routes to a IPv4 RIB using the rpc-add route function in the I2RS RIB [I-D.ietf-i2rs-rib-data-model]. Step 1 shows the route being configured via netconf as a static route, and step 2 shows how this static route is installed in the intended configuration. Step 3 shows how this static route is installed in the applied configuration and the derived status "installed" is added to the routing devices route table. Step 4 shows how the I2RS Client 1 adds the same route with a different next hop. In this example, there is only one nexthop per route so the ephemeral route replaces static route configuration and is synchronously installed in the applied configuration. Due to the

installation, the "installed" state is recorded in the kernel and associated with the I2RS RIB route.

In step 5, I2RS client 2 adds the same route to the intended configuration with a different next hop which replaces the route added by I2RS client 1 because I2RS Client 2 has a higher priority that client 1.

In step 6, I2RS client 2 removes the route. and the I2RS client 1 is notified of the removal. The I2RS client 1 re-write the route with a nexthop of 192.11.1.2, and the applied configuration is updasted.

In step 7, the I2RS Client 1 removes route and the local configuration is restored in the intended configuration. The intended configuration sent to applied configuration as part of the restoration.



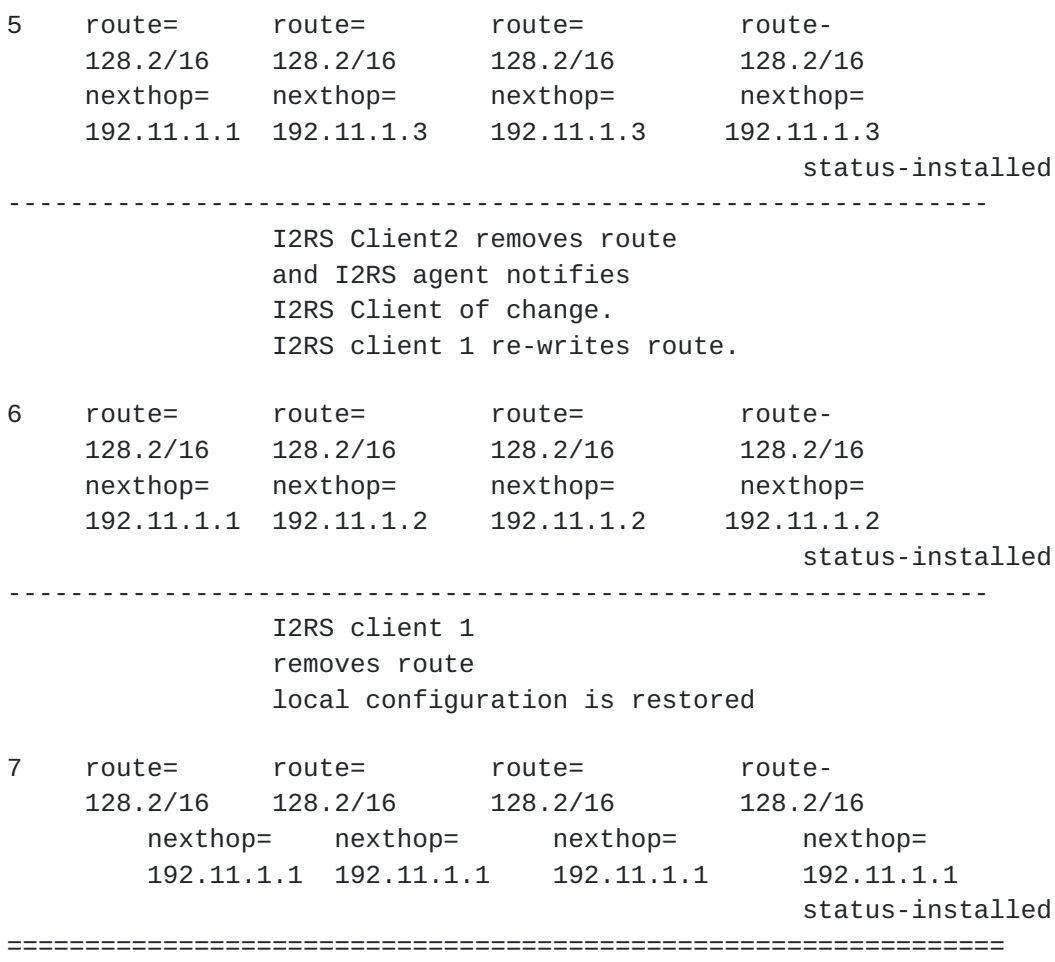


Figure 12

6.1. Portions of I2RS YANG data model

```
   module I2rs-RIB {
     ..
   module i2rs-rib {
        container routing-instance {
         ...
          list rib-list {
           ...
              list route-list {
                  key "route-index";
                  uses route;
             }
        }

     ....
      grouping route  {
           description
             "The common attribute used for all routes;"
            uses routeg-prefix;
             container nexthop {
                      uses nexthop;
                  }
              container route-statistics {
                  leaf route-state {
                     type route-state-def;
                     config false;    /* operational state */
                   }
                 leaf route-installed state {
                     type route-installed-state def;
                     config false;
                    }
                 leaf route-reason {
                   type route-reason-def;
                    config false;
                  }
              }
          container router-attributes {
               uses router-attributes;
            }
         container route-vendor-attributes
              uses route-vendor attributes;
        }
     }
```

Figure 13 - Simplified I2RS Route Model

6.2. NETCONF Changes

The NETCONF way of writing the ephemeral I2RS data would be:

(TBD)

Figure 14

6.3. RESTCONF Changes

Figure 8 shows the thermostat model has ephemeral variable desired-temp in the running configuration and the ephemeral data store. The RESTCONF way of addressing is below:

RESTCONF ephemeral datastore

(TBD)

Figure 15 - RESTCONF Route change

7. Previously Considered Ideas

7.1. A Separate Ephemeral Data store

The primary advantage of a fully separate data store is that the semantics of its contents are always clearly ephemeral. It also provides strong segregation of I2RS configuration and operational state from the rest of the system within the network element.

The most obvious disadvantage of such a fully separate data store is that interaction with the network element's operational or configuration state becomes significantly more difficult. As an example, a BGP I2RS use case would be the dynamic instantiation of a BGP peer. While it is readily possible to re-use any defined groupings from an IETF-standardized BGP module in such an I2RS ephemeral data store's modules, one cannot currently reference state from one data store to another

For example, XPath queries are done in the context document of the data store in question and thus it is impossible for an I2RS model to fulfil a "must" or "when" requirement in the BGP module in the standard data stores. To implement such a mechanism would require appropriate semantics for XPath.

7.2. Panes of Glass/Overlay

I2RS ephemeral configuration state is generally expected to be disjoint from persistent configuration. In some cases, extending persistent configuration with ephemeral attributes is expected to be useful. A case that is considered potentially useful but problematic was explored was the ability to "overlay" persistent configuration with ephemeral configuration.

In this overlay scenario, persistent configuration that was not shadowed by ephemeral configuration could be "read through".

There were two perceived disadvantages to this mechanism:

The general complexity with managing the overlay mechanism itself.

Consistency issues with validation should the ephemeral state be lost, perhaps on reboot. In such a case, the previously shadowed persistent state may no longer validate.

8. IANA Considerations

This is a protocol strawman - nothing is going to IANA.

9. Security Considerations

The security requirements for the I2RS protocol are covered in [I-D.ietf-i2rs-protocol-security-requirements]. The security environment the I2RS protocol is covered in [I-D.ietf-i2rs-security-environment-reqs]. Any person implementing or deploying the I2RS protocol should consider both security requirements.

10. 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
- Ignas Bagdonas
- Andy Bierman
- Alex Clemm

- Eric Voit
- Kent Watsen
- Jeff Haas
- Keyur Patel
- Hariharan Ananthakrishnan
- Dean Bogdanavich
- Anu Nair
- Juergen Schoenwaelder
- Kent Watsen

11. Major Contributors

- Andy Bierman (Yuman Networks) - andy@yumaworks.com
- Kent Watson (Juniper) (kwatsent@juniper.net)
- Russ White (Linkedin)

12. List of I2RS Requirements

The I2RS protocol requirements which include requirements for:

- ephemeral state ([I-D.ietf-i2rs-ephemeral-state]),
- protocol security ([I-D.ietf-i2rs-protocol-security-requirements]),
- traceability ([I-D.ietf-i2rs-traceability]
- publication and subscription service ([I-D.ietf-i2rs-pub-sub-requirements],

The I2RS environment requirments are find on [I-D.ietf-i2rs-security-environment-reqs].

13. References

13.1. Normative References:

[I-D.kwatsen-netmod-opstate]
Watsen, K., Bierman, A., Bjorklund, M., and J. Schoenwaelder, "Operational State Enhancements for YANG, NETCONF, and RESTCONF", draft-kwatsen-netmod-opstate-02 (work in progress), February 2016.

[I-D.schoenw-netmod-revised-datastores]
Schoenwaelder, J., "A Revised Conceptual Model for YANG Datastores", draft-schoenw-netmod-revised-datastores-01 (work in progress), June 2016.

[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>.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005, <http://www.rfc-editor.org/info/rfc4107>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://www.rfc-editor.org/info/rfc4960>.

[RFC5339] Le Roux, JL., Ed. and D. Papadimitriou, Ed., "Evaluation of Existing GMPLS Protocols against Multi-Layer and Multi-Region Networks (MLN/MRN)", RFC 5339, DOI 10.17487/RFC5339, September 2008, <http://www.rfc-editor.org/info/rfc5339>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, DOI 10.17487/RFC5424, March 2009, <http://www.rfc-editor.org/info/rfc5424>.

[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>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <http://www.rfc-editor.org/info/rfc6242>.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <http://www.rfc-editor.org/info/rfc6244>.

[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>.

[RFC7158] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7158, DOI 10.17487/RFC7158, March 2014, <http://www.rfc-editor.org/info/rfc7158>.

[RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <http://www.rfc-editor.org/info/rfc7589>.

13.2. Informative References

[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-15 (work in progress), April 2016.

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-14 (work in progress), July 2016.

[I-D.ietf-i2rs-protocol-security-requirements]
Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-ietf-i2rs-protocol-security-requirements-06 (work in progress), May 2016.

[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-09 (work in progress), May 2016.

[I-D.ietf-i2rs-rib-data-model]
Wang, L., Ananthakrishnan, H., Chen, M., amit.dass@ericsson.com, a., Kini, S., and N. Bahadur, "A YANG Data Model for Routing Information Base (RIB)", draft-ietf-i2rs-rib-data-model-06 (work in progress), July 2016.

[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-09 (work in progress), July 2016.

[I-D.ietf-i2rs-security-environment-reqs]
Migault, D., Halpern, J., and S. Hares, "I2RS Environment Security Requirements", draft-ietf-i2rs-security-environment-reqs-01 (work in progress), April 2016.

[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-11 (work in progress), May 2016.

[I-D.ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-17 (work in progress), December 2015.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-15 (work in progress), July 2016.

[I-D.ietf-netconf-server-model]
Watsen, K. and J. Schoenwaelder, "NETCONF Server and RESTCONF Server Configuration Models", draft-ietf-netconf-server-model-09 (work in progress), March 2016.

[I-D.ietf-netconf-yang-library]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", draft-ietf-netconf-yang-library-06 (work in progress), April 2016.

[I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-10 (work in progress), July 2016.

[I-D.ietf-netconf-yang-push]
Clemm, A., Prieto, A., Voit, E., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-03 (work in progress), June 2016.

[I-D.ietf-netconf-zerotouch]
Watsen, K. and M. Abrahamsson, "Zero Touch Provisioning for NETCONF or RESTCONF based Management", draft-ietf-netconf-zerotouch-09 (work in progress), July 2016.

[I-D.ietf-netmod-opstate-reqs]
Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-04 (work in progress), January 2016.

[I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-14 (work in progress), June 2016.

[I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-02 (work in progress), July 2016.

[I-D.ietf-netmod-syslog-model]
Wildes, C. and K. Koushik, "Syslog YANG Model", draft-ietf-netmod-syslog-model-09 (work in progress), July 2016.

[I-D.ietf-netmod-yang-metadata]
Lhotka, L., "Defining and Using Metadata with YANG", draft-ietf-netmod-yang-metadata-07 (work in progress), March 2016.

[I-D.wilton-netmod-refined-datastores]
Wilton, R., "Refined YANG datastores", draft-wilton-netmod-refined-datastores-01 (work in progress), July 2016.

Authors' Addresses

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

Amit Daas
Ericsson

Email: amit.dass@ericsson.com