INTERNET-DRAFT
Intended Status: Proposed Standard
Expires: August 19, 2013

R. Fernando
P. Chinnakannan
M. Madhayyan
A. Clemm

February 15, 2013

YANG modifications for I2RS
draft-rfernando-i2rs-yang-mods-00

Abstract

Interface to Routing Systems (I2RS) provides the mechanisms for the programmatic access of routing system components. YANG [RFC 6020] is a modeling language that is used to specify an external visible data model of sub-system - for defining both configuration and operational data held in the networking device. NETCONF is the protocol that is used to perform these configurations and accessing the operational data.

This document proposes to use the YANG data modeling language for I2RS data models. It also proposes a set of YANG language changes to enable the I2RS data models for multi-client and programmatic access.

This document also proposes the scope for the I2RS programmed data set on the Routing System and the necessary mechanisms for data synchronization.

This document also recommends a binary encoding for "on-the-wire" transfer of the YANG data model elements instead of XML.

Status of this Memo

Table of Contents

1   Introduction

The Network Configuration Protocol (NETCONF) provides mechanisms to programmatically install, manipulate, and delete the configuration of network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data as well as the protocol messages. The NETCONF protocol uses a remote procedure call (RPC) paradigm and protocol operations are performed on top of this simple RPC layer. A client encodes an RPC in XML and sends it to a server using a secure, connection-oriented session. The server responds with a reply encoded in XML.

YANG is the NETCONF Data Modeling Language [RFC6020], which supports hierarchical modeling of data elements that represent the configuration and runtime state of a particular network element.

The elements defined by the YANG module can be used for NETCONF-based communication, including passing configuration and state data, remote procedure calls (RPCs), and notifications. Thus YANG allows one to completely describe all the data sent between a NETCONF client and server.

YANG models the hierarchical organization of data as a tree in which each node has a name and a value or a set of child nodes. It provides clear and concise descriptions of the nodes, as well the inter-relationship between those nodes.

Interface to routing system (I2RS) framework is a draft proposal that sets out to build a framework to provide a common, standard interface to allow programmatic access to the information maintained in a router, like the routing protocol states, statistics, Routing Information Base (RIB) states, interface and their states etc.

Fundamental to the I2RS is a clear data model that defines the semantics of the information that can be written and read. This document proposes the use of Yang to define I2RS data models. It further proposes a set of YANG language extensions in order satisfy all the requirements of I2RS.

Specifically, this document makes the following proposal:

o That I2RS shall use YANG for defining the data models that are applicable for I2RS. These models are termed 'I2RS data models' in this document.

o A mechanism to express data ownership for a data set injected by I2RS clients into a Routing System using the I2RS data models.

o A mechanism to express multi-client semantics and multi-client operations in an I2RS data model.

o A mechanism to express the life time scope of programmatic data injected by I2RS clients in a Routing System and to express them in the I2RS data models.

The draft proposal for binary encoding for "on-the-wire" transfer of the YANG data model elements instead of XML and the associated NETCONF version2 is specified in a companion document (draft-tbd). The extensions proposed in this document is independent of the other changes proposed and will operate with or without the binary encoding as well as NETCONF version 2.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2 I2RS Components

2.1 Overview

This section describes the roles performed by the different I2RS components when YANG is used as the modeling language and NETCONF used as the protocol.

2.2 I2RS components

An I2RS domain comprises of the following entities:

o A Routing System supporting the I2RS data models and interfaces.

o I2RS clients accessing and operating on the Routing systems using the I2RS interface.

o I2RS servers present within a Routing system and providing the interface into the I2RS services and associated data models.

o The NETCONF V2 Protocol that is used by the I2RS clients to access the I2RS services provided by the I2RS servers using a binary encoding format.

o The I2RS client admission and authorization control service, which is co-located within the routing system. This function is provided by

an external service and this document recommends using NACM model [RFC 6536].

3 I2RS extensions to Yang

3.1 Overview

I2RS clients operate on the I2RS data models to obtain the services provided by the Routing sub-systems. Client operations result in creation, modification and deletion of data sets in the I2RS data model tree, which is represented in YANG.

The data set injected by an I2RS client may be single valued, which is modeled as a "leaf" object or multi-valued, which are modeled as hierarchy of nodes using "containers", "lists", etc. The hierarchy of nodes, termed as "data model sub tree" or simply "sub tree", is conceptually owned by the I2RS client.

I2RS clients may access operational data and persistent configuration data. In this, they are like applications that automate management workflows. In addition, I2RS clients may inject ephemeral data. Ephemeral data corresponds to data that installs state, but that (unlike configuration data) is not persisted.

The ownership in this context requires identification of the I2RS client that injected the data set and the exclusivity assurance that the entire data set is injected and maintained by one client.

The exclusivity property guarantees that no other client shall modify any data item in the data set injected by one I2RS client. This concept of exclusivity assurance is proposed and described in section 3.2 Exclusive Owner, using the new YANG statement "exclusive-owner". That said, while an I2RS client is not able to modify the data that was injected by another client, it may able to view data injected by other clients (assuming proper authorization). This is required so that I2RS clients are able to understand the I2RS server's behavior.

In contrast, current NETCONF treats all data to be global, meaning modification of data by one client is visible to all other clients and can be modified by other clients as well. In short, NETCONF is inherently designed for multi-owner datastores. Whether data items can be modified or not depends on the intrinsic nature of the data item (specified by 'config' statement). The proposed YANG extensions allow establishing a concept of ownership to data that is based on the extrinsic property of who created it, rather than solely based on the intrinsic nature of the data item. The I2RS data models may be operated upon by multiple I2RS clients or a NETCONF client or a

mixture of both types of clients.

I2RS also requires a uniform and consistent mechanism to identify the client that had injected the data set for the entire data model or to the subset of data model. This is required so that ownership of data items can be tagged to the data item themselves. The YANG data model language must provide a capability to express and capture the identity of a data model client in uniform and consistent manner across all data models. The data model client identifier must be a canonical client name identifier whose value must reference an unique client name record in a NETCONF access control model, NACM, [RFC 6536] database within the domain. Section, 3.4 Canonical Client Name Identifier (cname) describes this capability through the canonical client name, "cname" extension to YANG.

The I2RS clients operations on routing sub-systems results in creation, modification and deletion of data elements that are represented by the I2RS data models. The I2RS data models not only provide the schema for the data set injected, but also indicate the sections within the data model that are modifiable or editable by an I2RS client. The term editable is used in this document to refer to any IRS operation that leads to creation, modification or deletion of data model elements.

The editable portions of a data model sub tree is indicated by the YANG statement "config true", and any resulting data set through edit operations are stored in a data store called the running configuration data store. This data set is functionally permanent, in the sense that this data set must be stored in a persistent data store by the Routing System component and it is present across router reboots.

However, in I2RS operations, one of the requirement is to allow the i2rs clients to perform edit operations and the resulting data set is not stored permanently. This data set is "ephemeral" and is lost once a router reboots. This requirement is met using a new YANG extension called "ephemeral true", which is described in section 3.5 Ephemeral data nodes.

An I2RS client injects a set of data items under a sub tree within an I2RS data model. The exclusivity assurance guarantees that no other client will be able to modify any data item in the data set injected by one I2RS client in a sub tree that is indicated through the YANG statement "exclusive-owner".

It is possible for I2RS clients to attach data items marked as "ephemeral" or "exclusive" as a subtree underneath another data node, defined in an existing YANG data module. The fact that the I2RS

client injects data under an existing subtree MUST NOT affect the behavior of this containing subtree. Specifically, the exclusivity property MUST NOT prevent an item that was previously configurable, from being locked for configuration purposes. If a container is deleted, any I2RS client data attached below it is deleted along with it. In practice, it is therefore RECOMMENDED that I2RS ephemeral data is only attached below subtrees that are not subject to configuration (e.g., data items representing an immutable property, such as the system as a whole), or highly unlikely to change (e.g., data items representing a physical interface).

Although 'exclusivity' and 'ephemeral' statements describe two different characteristics, in practice they tend to apply to the same set of data nodes. That said, the concept of exclusivity is orthogonal to the concept of ephemeral data. This means that the "exclusive" property can be applied to data nodes marked as "config true" just as it can be applied to data nodes that are marked as ephemeral. Applying an exclusivity concept to configuration data implies the need to apply this concept across datastores (running, startup, candidate etc). As noted before, NETCONF datastores are essentially global and do not carry the notion of ownership to data.

3.2 Exclusive Owner I2RS

Currently NETCONF/YANG allows an instance of a sub tree, created by one client, to be modified later by another client. A data model primitive is required that disallows modifications to ephemeral data elements under a sub tree created by one I2RS client by another I2RS client.

This mechanism provides the guarantee that no other client shall modify any data item in the data set injected by one I2RS client.

This draft specification proposes that a new YANG statement be supported that defines a data node as "exclusive". The presence of the statement indicates that the data node, and all its descendants, is modifiable only by the client which created the instance. Other clients can retrieve the data node, but cannot edit it or delete it. They also cannot add any of their own data nodes below it.

There are several scenarios for applying exclusive ownership. The following outlines some of them:

o Exclusive ownership shall apply for an entire list. This can be visualized as multiple identical tables, with each table being owned by one client. In this case, the YANG data model needs to be defined such that the list is contained in a container. The exclusive property is applied to that container.

o Exclusive ownership shall apply for a list element, but different list elements can have different owners. This can be visualized as a single table with multiple rows, with each row owned by a different client. In this case, the exclusive property is applied at the level of the definition of the list.

o Exclusive ownership shall apply only to a particular cell within a list element. This can be visualized as a single table with one cell of a row owned by a different clients. In this case, the exclusive property is applied at the level of the definition of the data node within the list.

The requirement for exclusive ownership is best illustrated with a few examples.

In the first example, the exclusive property is applied to a container within a list element. This means that within a list element, the container and everything it contains can only be modified by its owner. However, different list elements can have different owners for the corresponding container. Also, data nodes that are part of the same list element, but siblings of the container, can be modified by other clients as well.

```
   module i2rs-routing
   {

     imports ietf-i2rs-extensions {
             prefix "ix";
     }

      config true;

      description "An i2rs-routing module that contains sub tree nodes
      that are either single-owner or multi-owner. This example is for
      illustration of the single owner data node that represents a
      sub-section of cells in a row";

      list i2rs-route {
          key "prefix";
              .....

          leaf prefix {
          type inet:ip-prefix;
          }

          container i2rs-route-attributes {
              ix:exclusive;
              ix:ephemeral;
```

```
                //This allows the name of the owning client
                //to be returned as part of the instantiated
                //information
                uses ix:client;

                description "The i2rs-route-attributes container contains
                several properties of a route. Only the client that
                originally created the container is allowed to modify or
                delete the container and its contents.";
            }

            container i2rs-dont-care {
                description "The i2rs-don't-care container contains
                properties of a route that can be manipulated
                by any client";
             }
         }
     }
```

In the second example, the exclusive property is applied to list elements. Each list element has an exclusive owner. However, the same list can contain many list elements, and different list elements can have different exclusive owners.

```
     module i2rs-routing
     {
            imports ietf-i2rs-extensions {
                prefix "ix";
            }

            config true;

            description "An i2rs-routing module that contains sub tree
            nodes that are either single-owner or multi-owner. This
            example is for illustration of the single owner data node that
            represents a single row within a table";

            list i2rs-route {
                ix:exclusive;
                ix:ephemeral;
                key "prefix ix:cname";

                //This allows the name of the owning client
                //to be returned as part of the instantiated
                //information, and to be a part of the key
                uses ix:client;
```

```
                 container i2rs-route-attributes {
```

```
                      description
                      "The i2rs-route-attributes container contains
                       several properties of a route. Each list element has
                       an exclusive owner. Only the client that created the
                       list element can edit or delete it. In case of
                       multiple list elements with the same prefix, the
                       element that is associated with the client of the
                       highest priority takes effect. ";
               }
           }
       }
```

Note that in the above example, the client name ("ix:cname") is used as a key. This allows two separate clients, distinguished by their cname, to inject a list element of otherwise the same key.

Note furthermore that in this example, if there are multiple list elements that have the same prefix as part of their index, only a single list element (i.e. only a single route entry) for the same prefix will be in effect. This is resolved by virtue of the owning client's priority, as determined by the server.

This example addresses the following scenario: Consider two I2RS clients, "sdn-app-A" and "sdn-app-b", operating on the above data model and injecting routes. Both clients may inject the same prefix "10.0.0.0/8" and will result into two separate data nodes in the hosting routing system component. The data record introduced by the I2RS client "sdn-app-A" is identified by the keys {10.0.0.0/8, "sdn-app-A"}, which contains an instance of the container i2rs-route-property and an instance of the container i2rs-don't-care. Likewise, the data record introduced by the I2RS client "sdn-app-B" is identified by the keys {10.0.0.0/8, "sdn-app-B"}.

It would have been possible to declare the same list without the client name as a key. In that case, each list element still has its own exclusive owner. However, two clients cannot own a list element that otherwise has the same key.

In the third example, the exclusive property is applied to the entire list. In this case, a container for the list is introduced and the exclusive property applied at that level.

```
      module i2rs-routing
      {
             imports ietf-i2rs-extensions {
                 prefix "ix";
             }

             config true;

             description "An i2rs-routing module that contains sub tree
             nodes that are either single-owner or multi-owner. This
             example is for illustration of the single owner data node that
             represents a conceptual table";


             container i2rs-routes {

                 ix:exclusive;
                 ix:ephemeral;
                 uses ix:client;

                 description
                "This container serves as the container of a list of
                i2rs routes that has an exclusive owner.  Only the
                owner of this container can modify the list and any
                list elements within it.";

                 list i2rs-route {
                    key "prefix";

                    container i2rs-route-attributes {
                        description "The i2rs-route-attributes container
                        contains several properties of a route. ";
                }
            }
        }
```

It should be noted that the concept of exclusive ownership pertains only with regards to modification operations, not to retrieval operations. Any client can retrieve data, even if it is exclusively owned by another client.

3.3 Canonical Client Name Identifier (cname)

A routing system supports data models for different components within the system which are known as I2RS data models in this specification. The I2RS data models may be operated by multiple I2RS clients or a single configuration NETCONF client or a mixture of both types of

clients. This programmatic access requires the concept of a client and associated properties to be specified in the data model. A data model requires a uniform and consistent methodology, and a mechanism to identify the client that had injected the data set for the entire data model or to the subset of the data model. The YANG data model language must provide a capability to express and capture the identity of a data model client in uniform and consistent manner across all data models.

This specification proposes the following elements to meet with the I2RS framework:

A new data type "cname" is introduced, used to identify the client that owns a particular data node.

The data model client identifier value must reference a unique client name record in a NETCONF access control model, [RFC 6536] database within the domain. The NACM service within the domain shall be used for specifying the client access privileges and other authentication, authorization records.

A grouping, "client" is introduced which contains a client identifier of cname type. The grouping may be used in conjunction with any data nodes (e.g. containers or lists) whose data owner needs to be identified. The client identifier is always provided by the server, reflecting the client name as identified through NACM.

The client identifier may be used as key for identifying the client in the data model sub tree sections that requires unambiguous identity of the data owner.

This document specifies that I2RS data models use "cname" type to specify the client identifier at required sections that are specified using the "exclusive" keyword.

There are times when data provided by multiple clients need to be maintained in a list with clear ownership of nodes maintained in the list. For instance, next-hops for the same prefix could be provided by multiple applications.

This type of data model construct can be achieved in two ways as follows:

o Introducing an explicit YANG "list" node with a single key of the type "cname" as a parent node for a data model sub tree that is a non list type like containers, leaf-list, leaf etc.

o Introducing an additional key of the type "cname" for a data model

sub tree that is a YANG list.

```
   module i2rs-routing {

           imports ietf-i2rs-extensions {
               prefix "ix";
           }

           list i2rs-routing-client {
               key "ix:cname";
               ix:exclusive;

               description "The i2rs-routing-client is a list node
               that provides the client identifier of the I2RS client
               performing programmatic operation below this
               sub tree in the data model";

               uses ix:client;
```

```
                container i2rs-routing-table {

                    description "Routing table maintained per client";
               /* Contents not specified here*/
                }
         }
    }
```

The following code fragment demonstrates a I2RS programmatic sub tree that is a list node.

```
   module interfaces {

       imports ietf-i2rs-extensions {
           prefix "ix";
       }

       container  a-non-i2rs-container {
         /* Contents not specified */
       }

       list i2rs-routing-interface {
           key "ix:cname if-name";
           ix:exclusive;

           description "i2rs-routing-interface provides a list of i2rs
           interfaces. The I2RS client identifier performing
           programmatic access to the routing interfaces are identified
```

```
           by the key client-id.";

           uses ix:client;

           leaf if-name {
               type string;

               description "The key field which is of the string type
               that specifies the interface name";
           }
             /* Other nodes not specified */
```

```
          }
      }
```

3.4 Ephemeral data nodes

The I2RS clients operations on routing sub-systems results into creation, modification and deletion of data elements that are represented by the I2RS data models. The I2RS data models not only provide the schema for the data set injected, but also indicate the sections within the data model that are modifiable or editable by an I2RS client.

The editable portions of a data model sub tree is indicated by the YANG statement "config true", and any resulting data set through edit operations are stored in a data store called the running configuration data store. This data set is functionally permanent, in the sense that this data set must be stored in a persistent data store by the Routing System component and it is present across router reboots. However, in Routing System operations a requirement is present that must allow clients to perform edit operations and the resulting data set is not stored permanently. This data set is ephemeral and is present only for the duration in which the Router System component is up and operational. This specification calls this persistency nature of the I2RS client state data on the routing system as ephemeral state and the data associated as ephemeral state data.

NETCONF allow its clients to create, modify, and delete data model elements expressed in YANG. This data created by NETCONF operations is called as configuration data and is associated with a certain type of data store called running configuration. The data present in the running configuration data store is permanent and is present across routing system reboots and is in contrast with the state data created and maintained by I2RS clients. This requires a new YANG extension to designate ephemeral state data. It must be noted that only sections of the data model that use the YANG statement "config true" are

editable. Other sections of the data model that use the YANG statement "config false" are not editable and these type of data are called as operational data. Operational data is not associated with any data store. I2RS clients need a third type of data which must be

editable, but not persistent across routing system component reboots
as well as not associated with any data store like the operational
data.

This specification proposes a new YANG statement called "ephemeral"
be supported which takes one argument that is a string with the value
that "true" or "false". A value of "true" indicates the inclusive
data node and all its sub tree is editable and remains persistent as
long as the routing system component is up and operational. Value of
'false' serves the same purpose of 'config false' i.e. to mark a
schema node as operational data.

The YANG statement "ephemeral true;" may be a specified at any node
of the data model tree and is inherited by all descendant nodes.

Ephemeral data cannot be part of a startup or candidate datastore.
It can only be part of a running datastore.

The following example shows a valid use of the "ephemeral" statement.

```
   module i2rs-routing {

     description "An i2rs-routing module with the root node of
     designated as configurable for illustration purpose";

        container i2rs-routing-main-container {

         config true;


         container i2rs-section {
                   ix:ephemeral;

                   description "The node i2rs-section and its entire sub
                   tree is now made ephemeral. This indicates that this
                   node and all children of the container i2rs-section is
                   writable but not configurable any more. Any writes
                   made to this sub tree is not present in the running
                   configuration and is lost when the router reboots";
            }

            container config-section {
                   config true;
```

```
                      description "The config-section node and its sub tree
                      is editable and is stored in the running config data
                      store. This sub tree is stored persistently across
                      reboots";
               }

               container oper-section {
                      config false;

                      description "This is node and its sub tree is not
                      editable and is the operational data";
               }

               container an-ephemeral-node {
                      ephemeral true;

                      container incorrect-config-node {
                        config true;

                        description "This node is marked incorrectly
                        as configurable and this is not permitted. A
                        descendant node of an ephemeral schema node cannot
                        be marked as 'config true'";
                     }
               }
          }
      }
```

The following example shows how a section of the ephemeral sub tree can represent operational data.

```
       module i2rs-routing {
         description "An i2rs-routing module with the root node of
         designated as configurable for illustration purpose";

            container i2rs-routing-main-container {
             config true;

             container another-ephemeral-node {
                    ix:ephemeral;

                    container an-intermediate-oper-node {
                   config false;

                        description "This node illustrates how an
```

```
                            ephemeral sub tree can have operational data.";
```

```
                 }
             }
            }
       }
```

The following example shows in correct usage of the ephemeral statement that would be flagged as error by the YANG compiler.

```
      module i2rs-routing {
          description "An i2rs-routing module with the root node of
          designated as configurable for illustration purpose";

          container i2rs-routing-main-container {
              config true;

              container an-ephemeral-node {
                  ix:ephemeral;

                  container incorrect-config-node {
                      config true;

                       description "This node is marked incorrectly
                       as configurable and this is not permitted";
                  }
              }

              container an-oper-node {
                  config false;

                  container an-errored-ephemeral-node {
                      ix:ephemeral;

                        description "This node is incorrect and will
                        not be accepted by the data model development
                  }
              }
          }
      }
```

3.5 YANG module "ietf-i2rs-extensions.yang"

Yang module "ietf-i2rs-extensions" contains a set of definitions needed by clients that intend to inject ephemeral state into a device. Specifically, this module defines the following:

o A YANG extension that allows to declare the "ephemeral" clause

o A YANG extension that allows to declare the "exclusive" clause

o A grouping to be used in conjunction with data nodes that use the ephemeral clause. This grouping contains node to identify the owning control client and that client's priority.

o A set of management information containing a registry of control clients, expiration timers, and operational status (TBD)

```
file "ietf-i2rs-extensions.yang"

module ietf-i2rs-extensions {

    namespace "urn:ietf:params:xml:ns:yang:ietf-i2rs-extensions ";

    prefix "i2rs-extn";

    imports nacm {
        prefix nacm;
    }

    organization
        "example.com";

    contact
        "tbd";

    description
        "This module contains YANG extensions that would be needed to
         model i2rs specific data models.";

    revision "2013-02-14";

    extension ephemeral {
            description
            "This extension can be used on schema nodes to indicate that
            those data nodes and their descendant nodes do not survive
            device reload."

                argument access-specifier {
                    yin-element true;
                };
        }

        extension exclusive {
              description
              "This extension can be used on schema nodes to indicate that
              those data nodes and their descendant nodes can be created by
              and written to by only one i2rs client. Other clients may have
```

```
              have read-only access to those data node.;
        }

        typedef cname {
              type leafref {
                    path "/nacm:nacm/nacm:groups/nacm:group/nacm:name";
              }
              description
              "Control client ID by which the client is authenticated by the
              server.  An object of this type is always populated by the
              server, not by the invoking application.";
        }


        grouping client {
            leaf cname {
                config false;
                type cname;
            }
        }
    }
```

3.6 Client priority

Multiple I2RS clients can inject ephemeral state in a device. With the exclusivity statement in the data model, the state injected by multiple clients can all be maintained concurrently on the device.

Applications in the device that are interested in ephemeral state injected by multiple clients can subscribe to receive notifications and read from the data store. In some cases, I2RS clients can inject conflicting information. For instance, we could have multiple I2RS clients programming different next- hop for the same destination prefix.

In some cases, the backend applications would need to prioritize one entry over the other. To do so, we may need some mechanism to assign a relative priority to each of the i2rs client.

The proposal is to assign a priority to each client in the NACM database. In I2RS datamodels that use the 'cname', backend applications can receive the priority setting along with the client-id.

To set the priority for a NACM user, the proposal is to enhance the NACM user-group entries with a priority setting. Here is the corresponding YANG snippet, to be incorporated into a corresponding

YANG module:

```
augment /nacm:nacm/nacm:groups/nacm:group {

    leaf priority {
        type uint32;

        description
        "Relative priority setting of a user-group (to which a i2rs
         client gets assigned to".

        default 0;
    }
}
```

Both the user-group and priority setting is passed to the backend application if the data model uses the 'exclusive true' keyword. It is entirely up to the backend application to act on this data.

4 Security Considerations

This draft does not change the security characteristics of YANG.

5 IANA Considerations

This draft does not have any IANA considerations.

6 References

6.1 Normative References

[IRS-FRMWK] A. Atlas, T. Nadeau, D. Ward, "Interface to the Routing System Framework", draft-ward-irs-framework-00

[IRS-FRMWK-REQ] R. Fernando et al, "IRS Framework Requirements", draft-rfernando-irs-framework-requirement-00

7 Authors' Addresses

Rex Fernando
Cisco Systems
170 Tasman Dr
San Jose
EMail: rex@cisco.com

Palani Chinnakannan
Cisco Systems
170 Tasman Dr
San Jose
EMail: pals@cisco.com

Muthumayan Madhayyan
Cisco Systems
170 Tasman Dr
San Jose
EMail: muthu@cisco.com

Alexander Clemm
Cisco Systems
170 Tasman Dr
San Jose

EMail: alex@cisco.com