

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 23, 2014

R. Fernando
J. Medved
Cisco Systems
E. Crabbe
Google
K. Patel
Cisco Systems
October 20, 2013

I2RS Protocol Requirements
draft-rfernando-i2rs-protocol-requirements-00

Abstract

The Interface to Routing System (I2RS) allows an application to programmatically query and modify the state of the network. This document defines requirements for an I2RS protocol between applications (clients) and network elements (servers).

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 [[RFC2119](#)].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 23, 2014.

Internet-Draft

I2RS Protocol Requirements

October 2013

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
2.	Terminology	3
3.	I2RS Protocol High Level Design Objectives	5
4.	I2RS Protocol Requirements	7
4.1.	General Assumptions	7
4.2.	Transport Requirements	8
4.3.	Identity Requirements	10
4.4.	Message Encoding Requirements	11
4.5.	Message Exchange Pattern Requirements	11
4.6.	API Method Requirements	13
4.7.	Service and SDM Requirements	14
4.8.	Security Requirements	16
4.9.	Performance and Scale Requirements	17
4.10.	High Availability Requirements	18
4.11.	Application Programmability Requirements	19
4.12.	Operational Requirements	19
5.	Contributing Authors	20
6.	References	20
6.1.	Normative References	20
6.2.	Informative References	20
	Authors' Addresses	21

[1.](#) Introduction

I2RS defines a standard, programmatic interface for state transfer in and out of the Internet's routing system. I2RS is intended to automate network operations - it will allow applications to quickly interact with routing systems and to implement more complex operations, such as policy-based controls.

[[I-D.ietf-i2rs-problem-statement](#)] gives detailed problem statement for I2RS, while [[I-D.ietf-i2rs-architecture](#)] defines its high-level architecture.

This document expands on the required aspects of a protocol for I2RS, described in Section 5 of [[I-D.ietf-i2rs-problem-statement](#)], and refines I2RS protocol requirements formulated in Section 6 of [[I-D.ietf-i2rs-architecture](#)]. The purpose of this document is as follows:

1. To help stakeholders (equipment vendors, operators, application programmers or interested IETF participants), to arrive at a common understanding of the critical characteristics of the I2RS protocol(s).
2. To provide requirements to designers of the I2RS framework and protocols on aspects of the framework that warrant significant consideration during the design process.
3. To allow the stakeholders to evaluate technology choices that are suitable for I2RS, to identify gaps in these technologies and to help evolve them to suite I2RS's needs.

[2.](#) Terminology

This document uses the following terminology definitions.

Service: For the purposes of I2RS, a service refers to a set of related state access functions together with the policies that control it's usage. For example, the 'RIB service' is a service that provides access to state held in a device's RIB.

Server: Is a system that implements one or more services that can be accessed by client systems via well defined interfaces (APIs). A server can export multiple services. A server is typically a network device. [ed: as an aside, I still find this usage to be completely counterintuitive]

Client: A system that uses a service implemented by a server through a well defined interface. A client can make use of multiple services from multiple servers.

Participants: The server and client are collectively referred to as the participants of a service.

Transport: Any end-to-end mode of communication between a server and a client that allows exchange data the exchange of data. In principle, the transport hides the topology and other network properties from the participants of a service.

Messages: Messages are logical units of data that are exchanged between service participants.

Message Exchange Pattern: A categorization of a way in which messages could be exchanged between service participants. MEPs specify the sequence, order, direction and cardinality of messages exchanged. Request-response and asynchronous notifications are examples of MEPs.

Message Data Model: The schema representing the structure of messages being exchanged between the service participants. The MDMs can specify certain constraints such as data type, length, format and valid values of fields in messages.

Message Encoding: The "wire" representation of messages exchanged between service participants.

API Method: Is an application level procedure or a function that is invoked by the client to query or modify the state held in the server.

Service Scope: Is the functional scope of a service. The service scope is established during the service definition phase.
[definition needs work]

Service Data Model: The schema representing the conceptual structure of the state held in the server for a given service. The SDMs can specify certain constraints such as the data type, length, format and allowed values for fields representing the state. They also describe the relationship between various state elements that constitute the state.

Modeling Language: A language that defines schema for Message Data Models and Service Data Models.

Namespaces: A method for uniquely identifying and scoping of schemas declared for messages and services. Namespace is an important consideration when defining services and messages.

Service State or State: Is the general data held by the server for a given service.

State Element: A readable or writable state present in the server. The term 'State Element' may refer to states at different levels of granularity.

State Identifier: A unique identity for the state element. The identifier is derived from the SDM and uses the same naming convention as the SDM. State Identifiers may be viewed as the 'key' for the state.

State Value or 'value': A value that is assigned to a particular state identifier (key). The state is referred to using the State Identifier or 'key' in operations that sets or transfers the value of the state.

State Owner: Identity of the client that was the source of a state held in the server.

State lifetime: The duration which the state is maintained in the server.

Datastore: The physical mechanism used to store a service's state.

Capabilities: Capabilities represents the functionality supported by a server including the services supported and exported to clients.

Authentication: A mechanism that allows a server to recognize the identity of a client.

Authorization: A mechanism that allows determination of what an authenticated client is allowed to do.

Confidentiality: Specifies if data remains confidential while in transit between service participants.

Policy: For the purposes of this document, a policy is an explicit user configurable modification to the default behavior of the system. The enforcement could be conditional; a given policy could potentially become active only when certain conditions are met.

[3.](#) I2RS Protocol High Level Design Objectives

The core guiding principles and objectives that should be used in defining the specifics of the I2RS protocol(s) are as follows:

HLO-1. Design for Scale and Performance: is a key criteria for making design choices. The design should meet current and future performance and scale needs. Design patterns that allow us to compose a scalable, high performing system are well understood and should be utilized where possible.

HLO-2. Extensible: I2RS will be deployed in environments that will evolve over time. Hence the system should be designed with provisions that will allow significant modifications/extensions to be added to existing mechanisms. An extensible and future-proof design will drive better adoption as it is a promise against future technology churn.

HLO-3. Promote Reuse: Reuse in this context refers to using existing tools, technologies and mechanisms instead of inventing them from

scratch. It also refers to reusing a network device's current set of capabilities that applications could harness without reinventing them from scratch.

HLO-4. Promote Portability: Portability refers to the ease with which software written for one device or environment can be moved to work seamlessly with another device or environment to achieve similar functionality. A fundamental requirement for I2RS is to achieve predictive and consistent behavior when applications are migrated from one platform or environment to another.

HLO-5. Security: I2RS may be deployed in environments where it might be subjected to threats and denial-of-service attacks that might cause intentional damage to the functioning of a network. This could be in the form of loss of service, degradation of performance, loss of confidentiality, etc. Therefore, the I2RS protocol must provide basic security mechanisms including but not limited to authentication, authorization, confidentiality along with sufficiently expressive policy requirements for each.

HLO-6. Separation of concerns: The components of the system should be decoupled from each other as much as possible to achieve clear separation of concerns. This modularity would allow for interchangeable design and implementation choices that address the individual components requirements.

HLO-7. Robustness: Robustness is the ability of a system to operate in the face of failures and errors. It is also its ability to correctly and predictably recover from such errors and to settle to a known state. Since applications that use the I2RS framework are remote and would be controlling the entire network, ensuring fault tolerance is an important consideration.

Most of these requirements cut across all the components of the system and hence should be kept in mind while designing each component and the system as a whole.

[4.](#) I2RS Protocol Requirements

This section is divided into multiple sub-sections, each dealing with a specific consideration of I2RS protocol design. As we list the

requirements under each subsection, we'll annotate each requirement with what high level objectives they meet. Additional reasoning is additionally provided where appropriate.

[4.1.](#) General Assumptions

This section captures the general, high level assumptions of the I2RS framework. The context for defining protocol requirements is shown in the following figure.

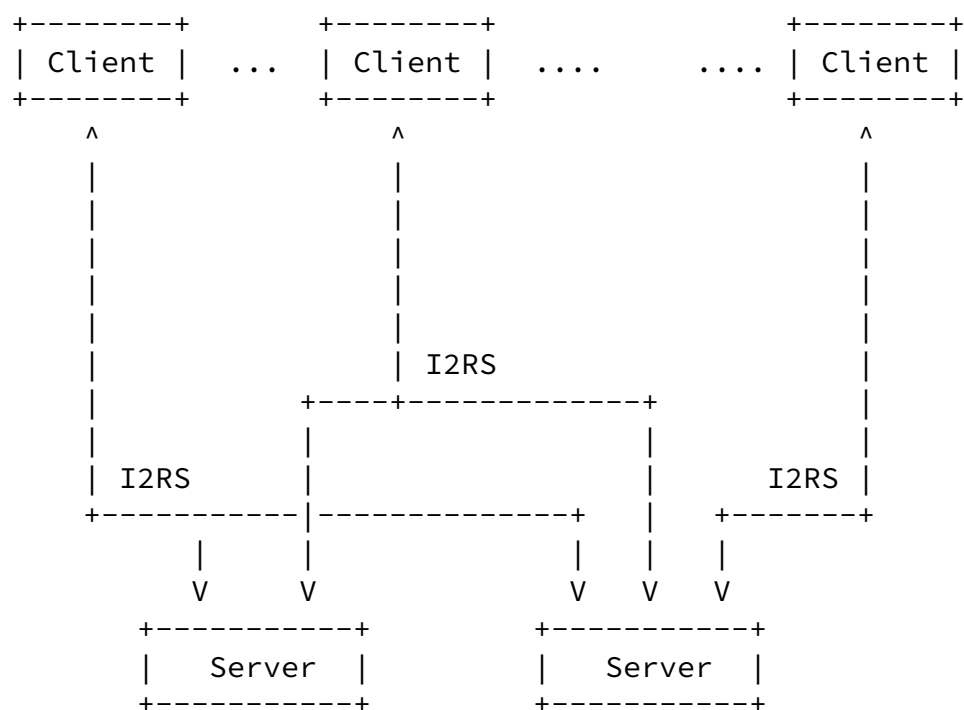


Figure 1: Protocol requirements context

Since the design choices for the I2RS framework are many, some simplifying assumptions could make the framework requirements more tangible and useful.

provided to an application by exposing a set of API's from the device to the application. Due to this characteristic, I2RS is a client-server protocol/framework. I2RS must provide mechanisms for the client to discover services that a server provides.

GEN-2: The client can use the APIs provided by the server to programmatically add, modify, delete and query state held in the server. Additionally clients can register for certain events and be notified when those events occur.

GEN-3: The client and the server communicate using a simple transport connection. The client initiates the transport connection to the server. The server does not know the number and timing of the connections from its clients.

GEN-4: A service provides access to the state held in the server structured according to the SDM of that service. A service allows a client the ability to manipulate the service state.

GEN-5: The I2RS MUST define a data model to describe the SDMs supported in the server and MUST define a data modeling language to formally describe that data model. I2RS MUST specify the mapping from the service data model to the message data model and subsequently to the client API.

[4.2.](#) Transport Requirements

The transport layer provides connectivity between the client and the server. This section details the transport requirements.

TR-1: There should exist a default transport connection between the client and the server for communication. This control connection is point-to-point and should provide in-order and reliable delivery of data in both directions. The simplest I2RS setup would only have a single transport session between the participants.

TR-2: Depending on the data being exchanged, there may be additional transport connections between the client and server defined in future. The characteristics of these additional transport connections will be dictated by the requirements that create them.

TR-3: The transport connection between the client and server MUST have mechanisms to support authentication, authorization and to optionally provide confidentiality of data exchanged between the client and the server. See [Section 4.8](#) for more details.

-
- TR-4: A client could connect to multiple servers. Similarly, a server could accept connections from multiple clients. Any participant may initiate the transport connection.
- TR-5: The exact technology used for the transport layer should be replaceable. There should be a single mandatory transport that must be supported by all participants. This requirement will ensure that there is always an interoperable transport mechanism between any client and any server.
- TR-6: Clients and servers by default communicate using a point-to-point transport connection.
- TR-7: Point-to-multipoint transports are mainly used to scale the system by avoiding ingress replication when the same message has to be delivered to multiple receivers. P2MP transport would work hand-in-hand with a P2MP MEP. The subject of P2MP transport and P2MP MEP is for future work.
- TR-8: Once the transport connection is established, it is desirable to reuse it to perform multiple operations. This requirement ensures that the system scales by amortizing the session setup cost across multiple operations. Session down events may not have an impact on the state maintained by the server.
- TR-9: After the transport is established, the participants exchange capabilities and other session parameters before exchanging service related messages.
- TR-10: Messages pertaining to multiple services could be exchanged over a single transport connection.
- TR-11: The "default" transport connection between a client and a server is purely for control plane message exchanges. Data plane packets are not expected to be sent over this "default" connection. When required, packets to be extracted from or injected to the data plane could be designed as a service in itself that sets up a separate packet injection/extraction channel that provides the correct characteristics.
- TR-12: For operational reasons, a participant MUST be able to detect a transport connection failure. To satisfy this requirement, transport level keep-alives could be used. If the underlying transport connection does not provide a keep-alive mechanism, it should be provided at the I2RS protocol level. For example, if TCP is used as a transport, TCP keep-alives could be used to

detect transport session failures.

[4.3.](#) Identity Requirements

I2RS could be used in a multi-domain distributed environment. Therefore a fool-proof way to ascertain the identity of clients is of utmost importance. Identity provides authenticated access to clients to state held by the server.

ID-1: Each client should have a unique identity that can be verified by the server. The authentication could be direct or through an identity broker.

ID-2: The server should use the client's identity to track state provided by the client. State ownership enables multiple clients to edit their shared state. This is useful for troubleshooting and during client death or disconnection when the client's state may have to be purged or delegated to another client that shares the same identity.

ID-3: The client's I2RS identity should be independent of the location or the network address of the physical node in which it is hosted. This allows the client to move between physical nodes. It also allows a standby client to take over when the primary fails and allows shared state editing by multiple clients as discussed in I.2.

ID-4: A client that reboots or reconnects after a disconnection MUST have the same I2RS identity if it wishes to continue to operate on the state that it previously injected.

ID-5: A clients ability to operate on a state held by the server is expressed at the granularity of a service. A service could be read-only or read-write by a client possessing a particular identity.

ID-6: A policy on the server could dictate the services that could be exposed to clients. Upon identity verification, the authorized services are exported to the client by capability announcement.

ID-7: A client can edit (write, delete) only the state that was

injected by it or other clients with the same shared identity. Therefore, two conditions must be met for a client to edit a state through a session. First, the client should receive capability from the server that it has 'edit' permissions for the service in question, and, secondly, the state that it edits should be its own state.

ID-8: The server retains the client's identity till all of that client's state is purged from the server.

[4.4.](#) Message Encoding Requirements

Clients and servers communicate by exchanging messages between them. Message encoding is the process of converting information content in a message to a form that can be transferred between them.

ME-1: Every message between the client and the server is encoded in a transport-independent frame format.

ME-2: Each message is serialized on the sender's side and de-serialized on the receiver's side. The technology used for encoding and decoding messages could be negotiated between the client and the server.

ME-3: A mandatory default encoding standard should be specified and implemented by all I2RS participants. This ensures that there is an interoperable default encoding mechanism between any client and any server.

ME-4: The mandatory encoding technology chosen should be well supported by a developer community and should be standards based. Availability of tools and language bindings should be one of the criteria used in selecting the mandatory encoding technology.

ME-5: If multiple message encoding is supported in the framework, the encoding used for the current session should be configured using a policy on the server side and negotiated using capabilities. Note that currently there is no requirement to support multiple encoding schemes.

ME-6: The message encoding standard should be language and platform neutral. It should provide tools to express individual fields in

a message in a platform independent IDL-based language.

ME-7: The encoding/decoding mechanism should be fast and efficient. It should allow for operation on legacy equipment.

ME-8: The encoding scheme should allow for optional fields and backward compatibility. It should be independent of the transport and the message exchange pattern used.

ME-9: Human readability of messages exchanged on the wire might be a goal but it is secondary to efficiency concerns.

[4.5.](#) Message Exchange Pattern Requirements

Message exchange patterns form the basis for all service level activities. MEPs create a pattern of message exchanges that any task

can be mapped to whether initiated by a client or the server. This section provides the requirements for MEPS.

MEP-1: I2RS defines three types of messages between the client and the server. First, capabilities need to be exchanged on session establishment. Second, API commands are sent from a client to a server to add, delete, modify and query state. And third, asynchronous notifications are sent from a server to a client when interesting state changes occur.

MEP-2: The above message exchanges can be satisfied by two message exchange patterns. Capabilities and asynchronous notifications can be satisfied by one-way unsolicited fire and forget message. API commands can be satisfied using a request-response message exchange. The base I2RS framework should thus support at least these two MEPs.

MEP-3: For a request-response MEP, the server SHOULD acknowledge every request message from the client with a response message.

MEP-4: The response message in a request-response MEP SHOULD indicate that the server has received the message, done some basic sanity checking on its contents and has accepted the message. The arrival of a response does not mean all post processing of the message has completed.

MEP-5: If an error occurs with an API command, The response message MUST indicate an error and carry error information if there was a failure to process the request. The error code SHOULD be accompanied by a descriptive reason for the failure.

MEP-6: Error codes SHOULD indicate to the client which layer generated that error (transport, message parsing, schema validation, application level failure, etc). The I2RS framework should specify a standard set of error codes.

MEP-7: The request-response messages SHOULD be asynchronous. That is, the client should not be required to stop-and-wait for one message to be acknowledged before it transmits the next request. [ed: there must be a method for dependency tracking in the protocol. possibly negotiate as an optional capability?]

MEP-8: To satisfy asynchronous operations, a mechanism MUST exist to correlate response messages to their respective original requests. For example, a message-id could be carried in the response that would help the sender to correlate the response message to its original request.

MEP-9: The response messages need not arrive in the order in which the request was transmitted. [ed: again, there must be a method for providing an order of operations, and for receiving positive ack on completion]

MEP-10: The request message SHOULD carry an application cookie that should be returned back to it in the corresponding response. [ed: provide further justification]

MEP-11: Besides the request-response MEP, there is a need for a fire and forget MEP. Asynchronous notifications from the server to the client could be carried using this MEP. Fire and forget MEPs can be used in both client-to-server and server-to-client directions.

MEP-12: Fire-and-forget messages MAY optionally be acknowledged.

MEP-13: The fire-and-forget MEP does not carry a message-id but it SHOULD carry a cookie that can be set by the sender and processed

by the receiver. The cookie could help the receiver of the message to use the message for its intended purpose.

[4.6.](#) API Method Requirements

API methods specify the exact operation that one participant intends to perform. This section outlines the requirements for API methods.

API-1: The I2RS framework SHOULD provide for a simple set of API methods, invoked from the client to the server. These methods should allow to add, modify, query and delete of state that the server maintains.

API-2: The I2RS framework should provide for three query methods, subscribe, unsubscribe, and one-time-query, that the client can use to express an interest or collect specific state changes or state from the server.

API-3: The API methods discussed in API-1 and API-2 should be transported in a request-response MEP from the client to the server.

API-4: The API framework SHOULD provide for a single notify method from the server to the client when interested state changes occur. The notification method SHOULD be transported in a fire-and-forget MEP from the server to the client.

API-5: The framework SHOULD define a set of base API methods for manipulating state. These SHOULD be generic and SHOULD not be service specific.

API-6: All API methods that affect the state in the server SHOULD be idempotent. That is, the final state on the server should be independent of the number of times a state change method with the same parameters was invoked by the client.

API-7: All API methods SHOULD support a batched mode for efficiency purposes. In this mode multiple state entries are transmitted in a single message with a single operation such as add, delete, etc. For methods described in A.1 and A.2 which elicit a response, the failure mechanism that is specific to a subset of state in the batch should be devised. The Notify method SHOULD also support a

batched mode.

API-8: Since the API methods are primarily oriented towards state transfer between the client and server, there SHOULD be a identifier (or a key) to uniquely identify the state being addressed.

API-9: API methods that refer to value of a particular state SHOULD carry the state identifier (key) as well as the its value. For instance, during a state add operation, both the identifier (key) and the value SHOULD be passed down from the client to the server.

API-10: Besides basic API methods common to all services, a server could support proprietary methods or service specific methods. The framework SHOULD specify a mechanism to express these methods and their semantics through a modeling language or otherwise. The ability to support additional API methods SHOULD be conveyed to the client through capability negotiation.

API-11: Transactions allow a set of operations to be completed atomically (all or nothing) and that the end result is consistent. Transactions MAY be required for some network applications. [ed: i'd like to remove this for the the first version. it's a bit of a canard']

[4.7.](#) Service and SDM Requirements

SVC-1: Each service is associated with a service data model that defines the type and structure of the state (data) pertaining to that service. I2RS MUST provide mechanisms to manage the state held in the server in accordance to the SDM.

SVC-2: The base I2RS API methods MUST allow a client to add, modify, query and delete state information.

SVC-3: Neither the transport or the MEP SHOULD have any bearing on the structure of the state being transferred. Each service module

in the server would be responsible for interpreting the structure of the state being transferred corresponding to the SDM.

SVC-4: A client, after proper identification, could operate on

multiple 'services' that are exported to it. A client could have read-only or read-write access to a given service. The availability of the service, valid entities and parameter ranges associated with same are expressed via the exchange of capability information with the client.

SVC-5: The arrangement and structure of state (SDM) SHOULD be expressed in a network friendly data modeling language. [ed: I have no idea what this means]

SVC-6: The data modeling language SHOULD have the ability to express one-to-one, one-to-many and hierarchical relationships between modelled entities.

SVC-7: The data modeling language MUST allow for a service data model to be extended beyond its initial definition. The language MUST be able to express mandatory and optional elements in SDMs. It SHOULD also have the ability to express exceptions for unsupported elements in the model.

SVC-8: For every service that it wishes to expose to a client, the server SHOULD send capabilities that indicate the existence and identity of the service data model, as well as valid states and parameter ranges, any exceptions to it and the optional features of the data model that it supports.

SVC-9: The data modeling language MUST be able to express a dependence of service data model on another SDM. It SHOULD also have the ability to express references to state elements in another service data model.

SVC-10: The modeling language MUST have the ability to express read/write and read-only constraints for state elements in an SDM. Readable state elements are populated and managed by the server and clients don't have the ability to write their value. Routing next-hops added by a client is an example of read-write state. Statistics associated with that next-hop is an example of read-only state.

SVC-11: Query and notification APIs MUST be able to carry both read-only as well as read-write state.

SVC-12: Besides specifying a SDM, a service SHOULD also specify the interesting state changes that clients can subscribe to for

notifications. The absence of such a specification SHOULD be interpreted as an implicit unavailability of any subscribable entities. [ed: terminology to be worked out here]

SVC-13: A client which is authenticated to access a service (either read-only or read-write) MAY subscribe to any subscribable state change events.

SVC-14: A subscribe method SHOULD optionally have a filter associated. This increases the efficiency by filtering out events that the client is not interested in. The notification filter should have the ability to express state identifiers and wildcards for values.

SVC-15: The base API operations MUST be generic and allow a client to operate on multiple services with the same set of methods. Each service dictates its own schema or SDM.

SVC-16: I2RS protocol SHOULD allow a server to export standard services as well as vendor proprietary services. A namespace scheme should be devised to recognize standard and proprietary services.

SVC-17: The server SHOULD indicate to the client the availability of infrastructure to manage the state that it maintains. This includes but is not limited to the availability of persistent store, the availability of timer to clean up state after a specified timeout, the ability to clean up state on the occurrence of an event, etc. Equipped with this information, the client is responsible for the lifetime of the state.

SVC-18: Each state SHOULD have a set of meta data associated with it. This includes the state's owner, the state's lifetime attributes, a creation and modification timestamp, etc. This information would aid in the debugging of the system. An authenticated client that is exposed to a service SHOULD also have access to the meta data associated with that service's state.

[4.8.](#) Security Requirements

This section lists security requirements for the I2RS protocol.

SEC-1: Every client MUST be authenticated and associated with an identity. A secure mechanism to uniquely identify a client such as certificates MUST be adopted.

SEC-2: Every client MUST have an authorized role whereby only

certain state can be accessed and only certain operations can be

performed by that client. To keep the model simple and applications portable, authorization **MUST** be at a per service level and not on individual state element level.

SEC-3: The framework **MUST** provide for information confidentiality and information integrity as options.

SEC-4: Every state maintained by the server **SHOULD** be tagged with the client's identity as well as meta-data to indicate last access and last modifications time-stamps. This ensures accountability and helps auditing the system.

SEC-5: Mechanisms to "hook" into third-party security infrastructure whenever possible **SHOULD** be provided in order to achieve security goals. Applications programmers **SHOULD** be kept free of security concerns and yet given a flexible, configurable and well integrated security model.

[4.9](#). Performance and Scale Requirements

Performance requirements are usually woven in with the functional requirements of a system. They feature in every decision made to fulfill the systems requirements. Performance and scale are a complex function of many things. Hence performance requirements cannot be precisely quantified by a single number. This section lays out some common sense guidelines that should be kept in mind while designing the system from a scale and performance standpoint.

PS-1: The request-response MEP **SHOULD** be asynchronous. This ensures that a system is not stuck waiting for a response and makes the entire system more responsive and enables concurrency between operations.

PS-2: When applicable, messages **SHOULD** carry application level cookies that enable an application to quickly lookup the context necessary to process a message. Management of cookies is the applications responsibility.

PS-3: The framework **SHOULD** allow for bulk operations which amortizes the communication and messaging costs.

PS-4: The transport protocol SHOULD provide for a binary encoding option for messages between the participants.

PS-5: The transport protocol MUST provide for both encrypted and non-encrypted transport between participants.

PS-6: The transport protocol MUST provide for message prioritization.

PS-7: Multiple operations should be completed using a single transport session whenever possible.

PS-8: The server MUST be kept stateless with respect to the number and location of each client.

PS-9: Filtered subscription MUST be supported for notifications.

PS-10: An efficient synchronization mechanism between a client and a server SHOULD be provided that avoids transferring all the state between them.

PS-11: Allow clients that perform infrequent operations to disconnect their transport connection without cleaning up their state.

PS-12: Create the basic necessary mechanisms in the framework and build everything else as a service if possible.

[4.10](#). High Availability Requirements

The ability of the system to withstand operational failures and function in a predictable manner is called availability. A few guidelines that are important are,

HA-1: A 'superuser' identity SHOULD be provided that is capable of changing security policies, clearing state and perform actions that override client initiated actions in the system.

HA-2: Session disconnection and client deaths MUST be handled

gracefully. They should have the least impact on the server system.

HA-3: Client connections and disconnections MUST be logged and provided as a well-known service to authenticated users.

HA-4: Clients MUST be notified of message processing and other errors through error codes in messages.

HA-5: A mechanism to gracefully terminate the session between the client and the server MUST be provided.

HA-6: A mechanism for authenticated clients to query the load attributes of the system, both instantaneous and running average MUST be provided as a service.

[4.11.](#) Application Programmability Requirements

The framework should pay particular attention the requirements of application programmers. A well written framework should improve the productivity of programmers and shorten the time to make an application. This section has some issues to consider when devising the framework from an applications standpoint.

PGM-1: A client programming framework SHOULD allow applications writers to focus on the app functionality rather than mechanisms required to communicate with the server.

PGM-2: The application once written to certain requirements should be portable to other identical environments. The framework should not have fine grained data access controls as this would lead to a poorly written application with portability issues.

PGM-3: The framework should be devised in a manner that it is possible to automate code generation and constraint checking in popular programming languages. Generated code can then be used readily by application programmers instead of dealing with the nitty-gritties of the system.

PGM-4: Define a common repository for SDMs from which clients can obtain the SDMs they are interested in and automatically generate most of the boilerplate code.

PGM-5: Provisions SHOULD be made for debugging and troubleshooting tools that includes message trace, call traces, access to relevant server traces and logs, packet decode tools to trace and decode messages on the wire, consistency checkers of state inserted into a server.

PGM-6: The toolset should have a general portion (for common functions, such as session management) and SDM-specific portions (for example, a flag to control generation of debug code in code generated for a particular SDM).

PGM-7: The framework SHOULD define SDMs and MDMs in a language neutral format so as to enable code generation in multiple programming languages.

[4.12.](#) Operational Requirements

OP-1: There is a need to identify operational performance parameters of the system and provide mechanisms to retrieve them from a running system.

OP-2: Provide a way to upgrade a service independently of the other services. This modularity allows uninterrupted operation of the all but one service which is being upgraded.

OP-3: Provide a detailed workflow for bringing about a new service. This workflow will start with the need to introduce a new service and address the following: How SDMs defined? Where are they standardized? How are new entities (MEPs, transport, encoding) introduced? What are the tools and workflow involved to develop and operationalize a service. The intent is to introduce a level of understanding about stakeholders responsibilities.

OP-4: Provide mechanisms and methodologies to test a new service before deployment.

[5.](#) Contributing Authors

Thanks to the following people for reviewing and providing meaningful contributions: Alia Atlas, Palani Chinnakannan, Alexander Clemm,

Muthumayan Mahdhayyan, John McDowell and Bruno Rijsman and David Ward.

[6.](#) References

[6.1.](#) Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.

[6.2.](#) 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-00](#) (work in progress), August 2013.

[I-D.ietf-i2rs-problem-statement]
Atlas, A., Nadeau, T., and D. Ward, "Interface to the Routing System Problem Statement", [draft-ietf-i2rs-problem-statement-00](#) (work in progress), August 2013.

[RFC1776] Crocker, S., "The Address is the Message", [RFC 1776](#), April 1995.

[RFC1925] Callon, R., "The Twelve Networking Truths", [RFC 1925](#), April 1996.

Fernando, et al.

Expires April 23, 2014

[Page 20]

Internet-Draft

I2RS Protocol Requirements

October 2013

Authors' Addresses

Rex Fernando
Cisco Systems
170 W Tasman Dr,
San Jose, CA 95134
US

Email: rex@cisco.com

Jan Medved

Cisco Systems
170 W Tasman Dr,
San Jose, CA 95134
US

Email: jmedved@cisco.com

Edward Crabbe
Google
1600 Amphitheater Parkway
Mountain View, CA 94043
US

Email: edward.crabbe@gmail.com

Keyur Patel
Cisco Systems
170 W Tasman Dr,
San Jose, CA 95134
US

Email: keyur@cisco.com