Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: May 14, 2015

M. Smith
R. Adams
M. Dvorkin
Cisco Systems, Inc.
Y. Laribi
Citrix
V. Pandey
IBM
P. Garg
Microsoft Corporation
N. Weidenbacher
Sungard Availability Services

November 10, 2014

OpFlex Control Protocol
draft-smith-opflex-01

Abstract

The OpFlex architecture provides a distributed control system based on a declarative policy information model. The policies are defined at a logically centralized policy repository (PR) and enforced within a set of distributed policy elements (PE). The PR communicates with the subordinate PEs using the OpFlex Control protocol. This protocol allows for bidirectional communication of policy, events, statistics, and faults. This document defines the OpFlex Control 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 May 14, 2015.

Table of Contents

1. Introduction

As software development processes merge with IT operations, there is an increasing demand for automation and agility within the IT infrastructure. Application deployment has been impeded due to the existing IT infrastructure operational models. Management at scale is a very difficult problem and existing imperative management models typically falter when challenged with the heterogeneity of various platforms, applications, and releases. In such environments, declarative management models have shown to cope quite well. In these systems, agents have autonomy of control and provide a declaration of intent regarding behavior. Declarative policy is rendered locally to provide desired system behavior. The OpFlex architecture is founded in these concepts.

1.1. 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].

1.2. Terminology

PD: Policy Domain. A logical instantiation of the OpFlex system components controlled by a single administrative policy.

EP: Endpoint. A device connected to the system.

OC: OpFlex Component. An entity that communicates using the OpFlex protocol.

EPR: Endpoint Registry. A logically centralized entity containing the endpoint registrations within associated policy domain.

OB: Observer. A logically centralized entity that serves as a repository for statistics, faults, and events.

PE: Policy Element. A function associated with entities comprising the policy policy domain that is responsible for local rendering of policy.

PR: Policy Repository. A logically centralized entity containing the definition of all policies governing the behavior of the associated policy domain.

OpFlex Device: Entity under the management of a Policy Element.

JSON: Javascript Object Notation [RFC4627]

2. Scope

This document defines the OpFlex Control Protocol used between OpFlex system components. It does not define the policy object model or the policy object model schemas. A System Overview section is provided for reference.

3. System Overview

OpFlex is a policy driven system used to control a large set of physical and virtual devices. The OpFlex system architecture consists of a number of logical components. These are the Policy Repository (PR), Endpoint Registry (EPR), Observer, and the Policy Elements (PE). These components and their interactions are described in the following subsections.

3.1. Policy Repository

Within each policy domain of the OpFlex system, there is a single logical entity referred to as the Policy Repository (PR) that serves as the single source of all policies. The PR handles policy resolution requests from the Policy Elements within the same policy domain. An example scope of an policy domain would be a datacenter fabric. These policies are configured directly by the user via a policy administration interface (API/UI/CLI/etc.) or indirectly (implicitly through the application of higher order policy constructs). These policies represent a declarative statement of desired state. Policies are typically abstracted from the underlying implementation.

3.1.1. Management Information Model

All of the physical and logical components that comprise the policy domain are represented in a hierarchical management information model (MIM), also referred to as the management information tree (MIT). The hierarchical structure starts at a root node and all policies within the system can be reached via parent and child containment relationships. Each node has a unique Uniform Resource Identifier (URI) [RFC3986] that indicates its place in the tree.

3.1.1.1. Managed Object

Each node in the tree represents a managed object (MO) or group of objects and contains its administrative state and operational state. An MO can represent a concrete object, such as a switch or adapter,

or a logical object, such as a policy or fault. An MO consists of the following items:

Properties: A property is a named instance of policy data and is interpreted by the Policy Element in local rendering of the policy.

Child Relations: A containment relationship between MOs where the children MOs are contained within the parent MO.

Parent Relation: The inverse of the children relationship. This relation is implicit and is implied through the hierarchical name of the MO name.

Observables: These are MOs that track state relevant to the observed MOs, such as statistics, faults, or health information. These MOs are reported to the Observer.

3.2. Endpoint Registry

The Endpoint Registry (EPR) is the component that stores the current operational state of the endpoints (EP) within the system. PEs register the EPs with the EPR upon EP attachment to the local device where the PE is resident. Upon EP detachment, the registration will be withdrawn. The EP registration information contains the scope of the EP such as the Tenant or logical network as well as location information such as the hypervisor where the EP resides, or other metadata and labels as required by the policy model. The EPR can be used by PEs to resolve the current EPR registrations as well as receive updates when the information changes.

3.3. Observer

The Observer serves as the monitoring subsystem that provides a detailed view of the system operational state and performance. It serves as a data repository for performance and state data that pertains to the devices under control. This could include information related to trending, forensics, and long-term visibility data such as statistics, events, and faults. Statistical data is reported to the Observer at expiration of reporting intervals and statistics will be rolled up for longer-term trend analysis.

3.4. Policy Element

Policy elements (PEs) are a policy enforcement entity that is part of the policy domain. Policy elements reside on physical or virtual devices that are subjected to policy control under a given policy

domain. In addition, a policy element might serve as a proxy for a device that lacks an embedded policy element. Policies are resolved with the PR using the OpFlex protocol. This protocol allows bidirectional communication, and allows the exchange of policy information. Policies are represented as managed object "sub-trees". Upon policy resolution, the PE renders the policy to the configuration of the underlying subsystem, and continuously performs health monitoring of the subsystem. PEs perform local corrective actions as needed for the enforcement of policies in its scope. The PE performs this enforcement continuously rather than only when the policy is changed, which is a departure from a more traditional orchestrated scheme. Operational transitions can also cause new or additional/incremental policy resolutions such as the attachment of new EPs to the corresponding device.

4. OpFlex Control Protocol

The OpFlex Control Protocol is used by OpFlex system components to communicate policy and operational data. This document describes a JSON format and uses JSON-RPC version 1.0 [JSON-RPC]. The JSON-RPC transport SHOULD be over TCP.

4.1. JSON Usage

The descriptions below use the following shorthand notations for JSON values. Terminology follows [RFC4627].

<string>:
A JSON string. Any Unicode string is allowed. Implementations SHOULD disallow null bytes.

<integer>:
A JSON number with an integer value, within the range -(2**63)...+(2**63)-1.

<json-value>:
Any JSON value.

<nonnull-json-value>:
Any JSON value except null.

<URI>:
A JSON string in the form of a Uniform Resource Identifier[RFC3986].

<status>:
An enumeration specifying one of the following set of strings: "created", "modified", or "deleted".

<role>:
An enumeration specifying one of the following set of strings: "policy_element", "observer", "policy_repository", or "endpoint_registry".

<mo>:
A JSON object with the following members:

```
"subject": <string>
"uri": <URI>
"properties": [{"name":<string>, "data": <json-value>}*]
"parent_subject": <string>
"parent_uri": <URI>
"parent_relation": <string>
"children": [<URI>*]
```

All of the members of the JSON object are REQUIRED. However, the corresponding value MAY consist of the empty set for all members except for "name". It is REQUIRED that the "name" be specified.

The "subject" provides the class of entity for which the declaration applies. The applicable object classes are dependent on the particular MIT.

The "uri" uniquely identifies the managed object within the scope of the policy domain and indicates its location within the MIT.

The "properties" holds a set of named policy data.

The "parent_subject" provides the class of entity for the parent of the managed object. This field is optional. If omitted, then the managed object is a root element.

The "parent_uri" the uri of the parent managed object. The parent URI MUST be a prefix of the URI of the child object. This field is optional. If omitted, then the managed object is a root element.

The "parent_relation" is the name of the relation from parent to child. This field is optional. If omitted, the parent relation is considered equal to the subject field.

The "children" identifies a set of MOs where each MO is considered a child of this particular MO.

4.2. RPC Methods

The following subsections describe the RPC methods that are supported. As described in the JSON-RPC 1.0 specification, each request comprises a string containing the name of the method, a (possibly null) array of parameters to pass to the method, and a request ID, which can be used to match the response to the request. Each response comprises a result object (non-null in the event of a successful invocation), an error object (non-null in the event of an error), and the ID of the matching request. More details on each method, its parameters, and its results are described below.

A Policy Element is configured with the connectivity information of at least one peer OpFlex Control Protocol participant. The connectivity information consists of the information necessary to establish the initial connection such as the IP address and wire encapsulation. A Policy Element MAY be configured with the connectivity information for one or more of the OpFlex logical components. A Policy Element MUST connect to each of the configured OpFlex logical components.

4.2.1. Error Responses

In the event of an error, the response contains an error object, and the response object is null or omitted. The error response is as follows:

```
{
    "error": {
        "code": <string>,
        "message": <string>,
        "trace": <json-value>,
        "data": <json-value>
    },
    "id": <nonnull-json-value>
}
```

The "code" parameter is an error code that indicates the type of failure. The code MUST be one of the following strings:

"ERROR"
: A generic error not covered by any of the following errors.

"EUNSUPPORTED"
: The request could not be fulfilled because it is not supported.

"ESTATE"
: The session is not in a state that allows the specified request.

"EPROTO"
The OpFlex protocol version is not supported.

"EDOMAIN"
The policy domains do not match.

An OpFlex component MUST be capable of receiving an error response containing a code that is not in the above list, in which case it MAY treat the code the same as "ERROR".

The "message" parameter is a human-readable error message that corresponds to the error code. The message field is optional, but it MAY contain additional details not conveyed by the code.

The "trace" parameter is a JSON object that MAY convey debug or trace information, such as a stack trace, in an implementation-dependent way. This parameter is optional.

The "data" parameter is a JSON object that can contain additional data related to the specific request. This parameter MUST be present if required by the request.

The "id" parameter is the same ID as was sent in the request.

4.2.2. Identity

This method identifies the participant to its peer in the protocol exchange and MUST be sent as the first OpFlex protocol method. The method indicates the transmitter's role and the policy domain to which it belongs. Upon receiving an Identity message, the response will contain the configured connectivity information that the participant is using to communicate with each of the OpFlex components. If the response receiver is a Policy Element and is not configured with connectivity information for certain OpFlex logical components, it SHOULD use the peer's connectivity information to establish communication with the OpFlex logical components that have not been locally configured.

If a request type other than Identity is received before the Identity message, then the OpFlex Component MUST response with an error response with the error code set to ESTATE.

The Identity request is as follows:

```
{
    "method": "send_identity",
    "params": [{
        "proto_version": "1.0",
        "name": <string>,
        "domain": <string>,
        "my_role": [<role>+]
    }],
    "id": <nonnull-json-value>
}
```

The "proto_version" is a string that represents the version of the OpFlex protocol. This is the fixed value "1.0".

The "name" is an identifier of the OpFlex Control Protocol participant that is unique within the policy domain.

The "domain" is a globally unique identifier indicating the policy domain that this participant exists.

The "my_role" states the particular OpFlex component contained within this participant. Since a participant may be capable of acting as more than 1 type of component, there may be multiple "my_role" parameters passed.

The response object is as follows:

```
{
    "result": {
        "name": <string>,
        "my_role": [<role>+],
        "domain": <string>
          [ {"role": <role>,
          "connectivity_info": <string>}* ]
    },
    "error": null,
    "id": same "id" as request
}
```

The "name" is the identifier of the OpFlex Control Protocol participant sending the response.

The "my_role" states the OpFlex component roles contained within the participant sending the response.

The "domain" is a globally unique identifier indicating the policy domain that the participant sending the response exists.

The "role" and associated "connectivity_info" give the reachability information (i.e. IP address or DNS name) and the role of the entity that the participant is communicating using the OpFlex Control Protocol. This information MAY be gleaned by a receiving participant to resolve reachability for various OpFlex components.

If the protocol version is not supported by the recipient of the Identity request, then it MUST reply with an error response with the "code" field set to "EPROTO". If the policy domain does not match the policy domain of the recipient, then it MUST reply with an error response with the "code" field set to "EDOMAIN".

4.2.3. Echo

The "echo" method can be used by OpFlex Control Protocol peers to verify the liveness of a connection. It MUST be implemented by all participants.

The request is as follows:

```
{
    "method": "echo",
    "params": [],
    "id": <nonnull-json-value>
}
```

The response object is as follows:

```
{
    "result": {},
    "id": same "id" as request
}
```

4.2.4. Policy Resolve

This method retrieves the policy associated with the given policy name. The policy is returned as a set of managed objects. This method is typically sent by the PE to the PR.

The request is as follows:

```
{
    "method": "policy_resolve",
    "params": [{
        "subject": <string>,
        "policy_uri": <URI>,
        "policy_ident": {
            "name": <string>,
            "context": <URI>
        }
        "data": <string>,
        "prr": <integer>
    }*],
    "id": <nonnull-json-value>
}
```

The "subject" provides the class of entity for which the policy is being resolved. The applicable object classes are dependent on the particular MIT.

The "policy_uri" is the URI of the policy that needs to be resolved. Exactly one of "policy_uri" and "policy_ident" MUST be set.

The "policy_ident" is an identifier for the policy that needs to be resolved. It contains a "context" which is a scope or namespace in which the policy should be resolved, and a "name" which is a name that uniquely identifies the policy within the context. A common example of a context would be the URI of a tenant object. Exactly one of "policy_uri" and "policy_ident" MUST be set.

The "data" provides additional opaque data that may be used to assist in the policy resolution. This parameter is optional.

The "prr" or Policy Refresh Rate provides the amount of time that a PE should use the policy as provided in the request. The <integer> indicates the time in seconds that the policy should be kept by the PE. A PE SHOULD issue another policy resolution request before the expiration of the prr timer if the PE still requires the policy. If the PE is unable to subsequently resolve the policy after the prr timer expires, the PE MAY continue to use the resolved policy. The PE SHOULD raise an alarm if the policy cannot be resolved after multiple attempts.

Note that a policy resolve request can contain more than one request.

Upon successful policy resolution, the response object is as follows:

```
{
    "result": {
        "policy": [<mo>*],
    },
    "id": same "id" as request
}
```

The "policy" parameter contains the managed objects that represent the resolved policy. This includes the requested object and all of its transitive children. These objects are used by the Policy Element to render and apply the local policy. If the requested policy is not currently known, then the policy response MAY be the empty array.

When a policy resolution request is received by an OpFlex component, it MAY reply with the requested policy in the response. It MAY also reply with an empty response, and send the requested policy later with a policy update.

If the requested policy or any of its children are modified, deleted, or created, before the expiration of the PRR, the policy repository MUST send a policy update that represents the policy modifications. If the PRR expires before a new policy resolve message is received, then the policy repository SHOULD stop sending the updates. Note that these updates MUST be sent even if the requested policy is unknown at the time of the resolve request.

4.2.5. Policy Unresolve

This method indicates that the policy element is no longer interested in updates to a particular policy. Upon receipt of this message, the policy repository SHOULD stop sending updates related to the indicated policy object.

The request is as follows:

```
{
    "method": "policy_unresolve",
    "params": [{
         "subject": <string>,
         "policy_uri": <URI>,
         "policy_ident": {
              "name": <string>,
              "context": <URI>
          }}*
    ],
    "id": <nonnull-json-value>
}
```

The "subject" provides the class of entity to which the request applies. The applicable object classes are dependent on the particular MIT.

The "policy_uri" is a URI to unresolve that corresponds to an earlier resolve request. Exactly one of "policy_uri" and "policy_ident" MUST be set for each unresolve request.

The "policy_ident" is an identifier to unresolve that corresponds to an earlier resolve request. Note that a policy that was resolved using the "policy_ident" field can only be unresolved in the same way. Exactly one of "policy_uri" and "policy_ident" MUST be set for each unresolve request.

Note that a policy that was resolved using the "policy_uri" or "policy_ident" field can only be unresolved in the same way. If a policy is resolved multiple times in different ways, then the policy repository MUST continue to provide updates until all unique resolutions are either unresolved or timed out.

Further note that a policy unresolve request can contain multiple requests in the params list.

Upon successful completion, the response object is as follows:

```
{
    "result": {},
    "id": same "id" as request
}
```

4.2.6. Policy Update

This method is sent to Policy Elements when there has been a change of policy definition for policies for which the Policy Element has requested resolution. Policy Updates will only be sent to Policy Element for which the policy refresh rate timer has not expired.

The Policy Update contains the following members:

```
{
    "method": "policy_update",
    "params": [{
        "replace": [<mo>*],
        "merge-children": [<mo>*],
        "delete": [<URI>*],
    }],
    "id": <nonnull-json-value>
}
```

The "replace" parameter contains a list of changed managed objects. These objects completely replace the managed objects specified. If the existing object has any child elements that do not appear in the specified object's child list, then these child elements MUST be deleted.

The "merge-children" parameter contains a list of objects that will replace the properties of any existing object, including unsetting any properties that are set in the existing object but not set in the specified object. Any children that are specified will added to the set of children already present, but no children will be deleted.

The "delete" parameter specifies a list of URIs that reference objects that no longer exist and should be deleted.

The response object is as follows:

```
{
    "result": {},
    "id": same "id" as request
}
```

4.2.7. Endpoint Declare

This method is used to indicate the attachment or modification of an endpoint. It is sent from the Policy Element to the Endpoint Registry.

The request is as follows:

```
{
    "method": "endpoint_declare",
    "params": [{
        "endpoint": [<MO>+],
        "prr": <integer>}+
    ],
    "id": <nonnull-json-value>
}
```

The "endpoint" parameter is a list of managed objects representing the endpoints to declare. The endpoint managed object will contain one or more identifiers that can be used to look up the endpoint with an endpoint resolve request.

The "prr" or Policy Refresh Rate provides provides the amount of time that the endpoint declaration will remain valid. The <integer> indicates the time in seconds that the endpoint declaration should be kept by the EPR. A PE SHOULD issue another endpoint declaration

before the expiration of the prr timer if the endpoint is to continue existing within the system.

Note that an endpoint declare request can contain more than one endpoint declaration.

The response object is as follows:

```
{
    "result": {},
    "id": same "id" as request
}
```

4.2.8. Endpoint Undeclare

This method is used to indicate the detachment of an endpoint. It is sent from the Policy Element to the Endpoint Registry.

The request is as follows:

```
{
    "method": "endpoint_undeclare",
    "params": [{"subject": <string>,
                "endpoint_uri": <URI>}+
    ],
    "id": <nonnull-json-value>
}
```

The "subject" provides the class of entity to which the declaration applies. This will typically be the class representing the endpoint. The applicable object classes are dependent on the particular MIT.

The "endpoint_uri" is used to identify the endpoint or endpoints that are being detached.

Note that an endpoint undeclare request can contain more than one endpoint undeclaration.

The response object is as follows:

```
{
    "result": {},
    "id": same "id" as request
}
```

4.2.9. Endpoint Resolve

This method resolves the registration of a particular EP from the EPR. The request is made using the identifiers of the endpoint. Since multiple identifiers may be used to uniquely identify a particular endpoint, there may be more than 1 endpoint returned in the reply if the identifiers presented do not uniquely specify the endpoint.

The request is as follows:

```
{
    "method": "endpoint_resolve",
    "params": [{
        "subject": <string>,
        "endpoint_uri": <URI>,
        "endpoint_ident": {
            "context": <URI>,
            "identifier": <string>
        },
        "prr": <integer>}+
    ],
    "id": <nonnull-json-value>
}
```

The "subject" provides the class of entity to which the request applies. This will typically be the class representing the endpoint. The applicable object classes are dependent on the particular MIT.

The "endpoint_uri" is the URI of the endpoint that needs to be resolved. Exactly one of "endpoint_uri" and "endpoint_ident" MUST be set.

The "endpoint_ident" is an identifier for the endpoint that needs to be resolved. It contains a "context" which is a scope or namespace in which the endpoint should be resolved, and a "identifier" which is a name that uniquely identifies the endpoint within the context. A common example of a context would be the URI of an IP namespace object, and an within this namespace would be an IP address. Exactly one of "endpoint_uri" and "endpoint_ident" MUST be set.

The "prr" or Policy Refresh Rate provides provides the amount of time that the endpoint information will remain valid. The <integer> indicates the time in seconds that the endpoint information should be kept by the PE. A PE SHOULD issue another endpoint request before the expiration of the prr timer if the communication is still required with the endpoint.

Note that the endpoint resolve request can contain multiple endpoints to resolve.

The response object contains the registrations of zero or more endpoints. Each endpoint contains the same information that was present in the original registration. The following members are present in the response:

The response object is as follows:

```
{
    "result": {
        "endpoint": [<mo>*],
    },
    "id": same "id" as request
}
```

The "endpoint" parameter contains the managed objects that represent the endpoint registrations. This includes the requested object and all of its transitive children. If the requested endpoint is not currently known, then the policy response MAY be the empty array.

When an endpoint resolution request is received by an OpFlex component, it MAY reply with the requested endpoint registration in the response. It MAY also reply with an empty response, and send the requested policy later with an endpoint update.

If the requested endpoint object or any of its children are modified before the expiration of the PRR, the endpoint repository MUST send an endpoint update that represents the endpoint modifications. If the PRR expires before a new endpoint resolve message is received, then the endpoint repository SHOULD stop sending the updates. Note that these updates MUST be sent even if the requested endpoint is unknown at the time of the resolve request.

4.2.10. Endpoint Unresolve

This method indicates that the policy element is no longer interested in updates to a particular endpoint. Upon receipt of this message, the policy repository SHOULD stop sending updates related to the indicated policy object.

The request is as follows:

```
{
    "method": "endpoint_unresolve",
    "params": [{
        "subject": <string>,
        "endpoint_uri": <URI>,
        "endpoint_ident": {
            "context": <URI>,
            "identifier": <string>
        }}+
    ],
    "id": <nonnull-json-value>
}
```

The "subject" provides the class of entity for which the policy is being resolved. The applicable object classes are dependent on the particular MIT.

The "endpoint_uri" is a URI to unresolve that corresponds to an earlier resolve request. Exactly one of "endpoint_uri" and "endpoint_ident" MUST be set for each unresolve request.

The "endpoint_ident" is an identifier to unresolve that corresponds to an earlier resolve request. Exactly one of "endpoint_uri" and "endpoint_ident" MUST be set for each unresolve request.

Note that an endpoint that was resolved using the "endpoint_uri" or "endpoint_ident" field can only be unresolved in the same way. If an endpoint is resolved multiple times in different ways, then the endpoint registry MUST continue to provide updates until all unique resolutions are either unresolved or timed out.

Note that an endpoint unresolve request can can contain multiple unresolve requests in the params list.

Upon successful completion, the response object is as follows:

```
{
    "result": {},
    "id": same "id" as request
}
```

4.2.11. Endpoint Update

This method is sent to Policy Elements by the EPR when there has been a change relating to the EP Declaration for an Endpoint that the Policy Element has requested. Policy Updates will only be sent to Policy Elements for which the Policy Refresh Rate timer timer for the Endpoint Request has not expired.

The Endpoint Update contains the following members:

```
{
    "method": "endpoint_update",
    "params": [{
        "replace": [<mo>*],
        "delete": [<URI>*],
    }],
    "id": <nonnull-json-value>
}
```

The "replace" parameter contains a list of changed managed objects. These objects completely replace the managed objects specified. If the existing object has any child elements that do not appear in the specified object's child list, then these child elements MUST be deleted.

The "delete" parameter specifies a list of URIs that reference objects that no longer exist and should be deleted.

The response object is as follows:

```
{
    "result": {},
    "id": same "id" as request
}
```

4.2.12. State Report

This method is sent by the Policy Element to the Observer. It provides fault, event, statistics, and health information in the form of managed objects.

The state report contains the following members:

```
{
    "method": "state_report",
    "params": [{
        "object": <URI>,
        "observable": [<mo>*]}+
    ],
    "id": <nonnull-json-value>
}
```

The "object" parameter is a URI that indicates the managed object to which this state report applies.

The "observable" parameter is a list of managed objects that will be updated in this state report. Each of these managed objects will completely replace any existing observable managed object with the same URI.

The response object is as follows:

```
{
    "result": {},
    "id": same "id" as request
}
```

5. IANA Considerations

A TCP port will be requested from IANA for the OpFlex Control Protocol.

6. Security Considerations

The OpFlex Control Protocol itself does not address authentication, integrity, and privacy of the communication between the various OpFlex components. In order to protect the communication, the OpFlex Control Protocol SHOULD be secured using Transport Layer Security (TLS) [RFC5246]. The distribution of credentials will vary depending on the deployment. In some deployments, existing secure channels can be used to distribute the credentials.

7. Acknowledgements

The authors would like to thank Vijay Chander, Mike Cohen, and Brad McConnell for their comments and contributions.

8. Normative References

[JSON-RPC] Kollhof, J., "JSON-RPC Specification, Version 1.0", January 2006, <http://json-rpc.org/wiki/specification>.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

Authors' Addresses

Michael Smith
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134
USA

Email: michsmit@cisco.com

Robert Edward Adams
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134
USA

Email: readams@readams.net

Mike Dvorkin
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134
USA

Email: midvorki@cisco.com

Youcef Laribi
Citrix
4988 Great America Parkway
Santa Clara, California 95054
USA

Email: Youcef.Laribi@citrix.com

Vijoy Pandey
IBM
4400 N First Street
San Jose, California 95134
USA

Email: vijoy.pandey@us.ibm.com

Pankaj Garg
Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
USA

Email: pankajg@microsoft.com

Nik Weidenbacher
Sungard Availability Services
Philadelphia, Pennsylvania
USA

Email: nik.weidenbacher@sungard.com