Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 1, 2018

B. Carpenter
Univ. of Auckland
B. Liu, Ed.
Huawei Technologies
W. Wang
X. Gong
BUPT University

June 30, 2017

Generic Autonomic Signaling Protocol Application Program Interface (GRASP API) draft-liu-anima-grasp-api-04

Abstract

This document specifies the application programming interface (API) of the Generic Autonomic Signaling Protocol (GRASP). The API is used for Autonomic Service Agents (ASA) calling the GRASP protocol module to exchange autonomic network messages with other ASAs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on January 1, 2018.

Copyright Notice

Table of Contents

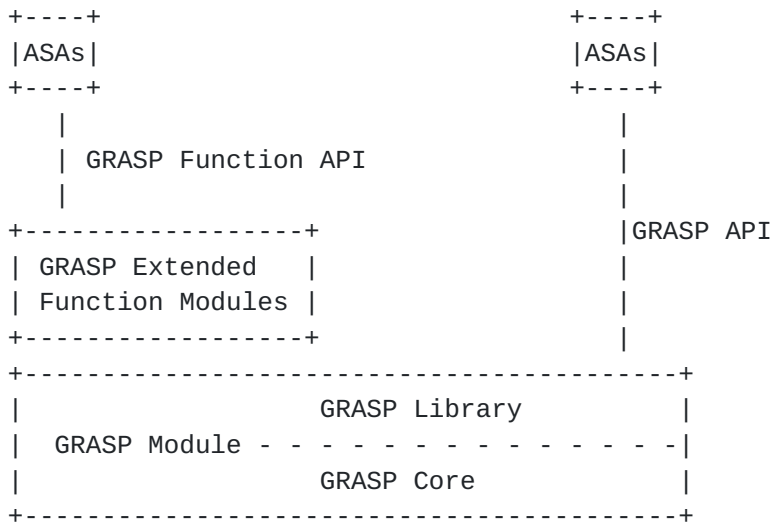
1. Introduction

As defined in [I-D.ietf-anima-reference-model], the Autonomic Service Agent (ASA) is the atomic entity of an autonomic function; and it is instantiated on autonomic nodes. When ASAs communicate with each other, they should use the Generic Autonomic Signaling Protocol (GRASP) [I-D.ietf-anima-grasp].

As the following figure shows, GRASP could contain two major sub-layers. The bottom is the GRASP base protocol module, which is only responsible for sending and receiving GRASP messages and maintaining shared data structures. The upper layer is some extended functions based upon GRASP basic protocol. For example, [I-D.liu-anima-grasp-distribution] describes a possible extended function.

It is desirable that ASAs can be designed as portable user-space programs using a portable API. In many operating systems, the GRASP module will therefore be split into two layers, one being a library that provides the API and the other being core code containing common

components such as multicast handling and the discovery cache. The details of this are system-dependent.



Both the GRASP base module and the extended function modules should be available to the ASAs. Thus, there needs to be two sub-sets of API. However, since the extended functions are expected to be added in an incremental manner, it is inappropriate to define the function APIs in a single document. This document only defines the base GRASP API.

Note that a very simple autonomic node might contain only a single ASA in addition to the autonomic infrastructure components described in [I-D.ietf-anima-bootstrapping-keyinfra] and [I-D.ietf-anima-autonomic-control-plane]. Such a node might directly integrate GRASP in its autonomic code and therefore not require this API to be installed.

2. GRASP API for ASA

2.1. Design Principles

The assumption of this document is that any Autonomic Service Agent (ASA) needs to call a GRASP module that handles protocol details (security, sending and listening for GRASP messages, waiting, caching discovery results, negotiation looping, sending and receiving sychronization data, etc.) but understands nothing about individual objectives. So this is a high level abstract API for use by ASAs. Individual language bindings should be defined in separate documents.

An assumption of this API is that ASAs may fall into various classes:

o ASAs that only use GRASP for discovery purposes.

- ASAs that use GRASP negotiation but only as an initiator (client).
- ASAs that use GRASP negotiation but only as a responder.
- ASAs that use GRASP negotiation as an initiator or responder.
- ASAs that use GRASP synchronization but only as an initiator (recipient).
- ASAs that use GRASP synchronization but only as a responder and/or flooder.
- ASAs that use GRASP synchronization as an initiator, responder and/or flooder.

The API also assumes that one ASA may support multiple objectives. Nothing prevents an ASA from supporting some objectives for synchronization and others for negotiation.

The API design assumes that the operating system and programming language provide a convenient mechanism for multi-threaded code. A solution in case this does not apply is described in Section 3.

This is a preliminary version. Two particular gaps exist:

- Authorization of ASAs is out of scope.
- The Rapid mode of GRASP is not supported.

2.2. API definition

2.2.1. Parameters and data structures

This section describes parameters and data structures uaed in multiple API calls.

2.2.1.1. Errorcode

All functions in the API have an unsigned 'errorcode' integer as their return value (the first returned value in languages that allow multiple returned parameters). An errorcode of zero indicates success. Any other value indicates failure of some kind. The first three errorcodes have special importance:

1. Declined: used to indicate that the other end has sent a GRASP Negotiation End message (M_END) with a Decline option (O_DECLINE).

2. No reply: used in non-blocking calls to indicate that the other end has sent no reply so far (see Section 3).

3. Unspecified error: used when no more specific error code applies.

Appendix A gives a full list of currently defined error codes.

2.2.1.2. Timeout

Wherever a 'timeout' parameter appears, it is an integer expressed in milliseconds. If it is zero, the GRASP default timeout (GRASP_DEF_TIMEOUT, see [I-D.ietf-anima-grasp]) will apply. If no response is received before the timeout expires, the call will fail unless otherwise noted.

2.2.1.3. Objective

An 'objective' parameter is a data structure with the following components:

- name (UTF-8 string) - the objective's name
- neg (Boolean) - True if objective supports negotiation (default False)
- synch (Boolean) - True if objective supports synchronization (default False)
- dry (Boolean) - True if objective also supports dry-run synchronization (default False)
  - Note 1: All objectives are assumed to support discovery, so there is no Boolean for that.
  - Note 2: Only one of 'synch' or 'neg' may be True.
  - Note 3: 'dry' must not be True unless 'neg' is also True.
- loop_count (integer) - Limit on negotiation steps etc. (default GRASP_DEF_LOOPCT, see [I-D.ietf-anima-grasp])
- value - a specific data structure expressing the value of the objective. The format is language dependent, with the constraint that it can be validly represented in CBOR (default integer = 0).

  An essential requirement for all language mappings and all implementations is that, regardless of what other options exist for a language-specific represenation of the value, there is

always an option to use a CBOR byte string as the value. The API will then wrap this byte string in CBOR Tag 24 for transmission via GRASP, and unwrap it after reception.

An example data structure definition for an objective in the C language is:

```
    typedef struct {
       char *name;
       bool neg;
       bool dry;
       bool synch;
       int loop_count;
       int value_size;           // size of value
       uint8_t cbor_value[];     // CBOR bytestring of value
       } objective;
```

An example data structure definition for an objective in the Python language is:

```
 class objective:
    """A GRASP objective"""
    def __init__(self, name):
        self.name = name    #Unique name, string
        self.neg = False    #Set True if objective supports negotiation
        self.dry = False    #Set True if objective also supports dry-run
negotiation
        self.synch = False  #Set True if objective supports synch
        self.loop_count = GRASP_DEF_LOOPCT  #Default starting value
        self.value = 0      #Place holder; any valid Python object
```

2.2.1.4. ASA_locator

An 'ASA_locator' parameter is a data structure with the following contents:

- locator - The actual locator, either an IP address or an ASCII string.
- ifi (integer) - The interface identifier index via which this was discovered - probably no use to a normal ASA
- expire (system dependent type) - The time on the local system clock when this locator will expire from the cache
- is_ipaddress (Boolean) - True if the locator is an IP address
- is_fqdn (Boolean) - True if the locator is an FQDN

- is_uri (Boolean) - True if the locator is a URI
- diverted (Boolean) - True if the locator was discovered via a Divert option
- protocol (integer) - Applicable transport protocol (IPPROTO_TCP or IPPROTO_UDP)
- port (integer) - Applicable port number

2.2.1.5. Tagged_objective

A 'tagged_objective' parameter is a data structure with the following contents:

- objective - An objective
- locator - The ASA_locator associated with the objective, or a null value.

2.2.1.6. Asa_nonce

In most calls, an 'asa_nonce' parameter is required. It is generated when an ASA registers with GRASP, and any call in which an invalid nonce is presented will fail. It is an up to 32-bit opaque value (for example represented as a uint32_t, depending on the language). It should be unpredictable; a possible implementation is to use the same mechanism that GRASP uses to generate Session IDs [I-D.ietf-anima-grasp]. Another possible implementation is to hash the name of the ASA with a locally defined secret key.

2.2.1.7. Session_nonce

In some calls, a 'session_nonce' parameter is required. This is an opaque data structure as far as the ASA is concerned, used to identify calls to the API as belonging to a specific GRASP session. In fully threaded implementations this parameter might not be needed, but it is included to act as a session handle if necessary. It will also allow GRASP to detect and ignore malicious calls or calls from timed-out sessions. A possible implementation is to form the nonce from the underlying GRASP Session ID and the source address of the session.

2.2.2. Registration

These functions are used to register an ASA and the objectives that it supports with the GRASP module. If an authorization model is added to GRASP, it would be added here.

- register_asa()

  Input parameter:

  name of the ASA (UTF-8 string)

  Return parameters:

  errorcode (integer)

  asa_nonce (integer) (if successful)

  This initialises state in the GRASP module for the calling entity (the ASA). In the case of success, an 'asa_nonce' is returned which the ASA must present in all subsequent calls. In the case of failure, the ASA has not been authorized and cannot operate.

- deregister_asa()

  Input parameters:

  asa_nonce (integer)

  name of the ASA (UTF-8 string)

  Return parameter:

  errorcode (integer)

  This removes all state in the GRASP module for the calling entity (the ASA), and deregisters any objectives it has registered. Note that these actions must also happen automatically if an ASA crashes.

  Note - the ASA name is strictly speaking redundant in this call, but is present for clarity.

- register_objective()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

  ttl (integer - default GRASP_DEF_TIMEOUT)

discoverable (Boolean - default False)

overlap (Boolean - default False)

local (Boolean - default False)

Return parameter:

errorcode (integer)

This registers an objective that this ASA supports and may modify. The 'objective' becomes a candidate for discovery. However, discovery responses should not be enabled until the ASA calls listen_negotiate() or listen_synchronize(), showing that it is able to act as a responder. The ASA may negotiate the objective or send synchronization or flood data. Registration is not needed if the ASA only wants to receive synchronization or flood data for the objective concerned.

The 'ttl' parameter is the valid lifetime (time to live) in milliseconds of any discovery response for this objective. The default value should be the GRASP default timeout (GRASP_DEF_TIMEOUT, see [I-D.ietf-anima-grasp]).

If the optional parameter 'discoverable' is True, the objective is immediately discoverable. This is intended for objectives that are only defined for GRASP discovery, and which do not support negotiation or synchronization.

If the optional parameter 'overlap' is True, more than one ASA may register this objective in the same GRASP instance.

If the optional parameter 'local' is True, discovery must return a link-local address. This feature is for objectives that must be restricted to the local link.

This call may be repeated for multiple objectives.

- deregister_objective()

Input parameters:

asa_nonce (integer)

objective (structure)

Return parameter:

errorcode (integer)

The 'objective' must have been registered by the calling ASA; if not, this call fails. Otherwise, it removes all state in the GRASP module for the given objective.

2.2.3. Discovery

- discover()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

  timeout (integer)

  flush (Boolean - default False)

  Return parameters:

  errorcode (integer)

  locator_list (structure)

  This returns a list of discovered 'ASA_locator's for the given objective. If the optional parameter 'flush' is True, any locally cached locators for the objective are deleted first. Otherwise, they are returned immediately. If not, GRASP discovery is performed, and all results obtained before the timeout expires are returned. If no results are obtained, an empty list is returned after the timeout. That is not an error condition.

  This should be called in a separate thread if asynchronous operation is required.

2.2.4. Negotiation

- request_negotiate()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

peer (ASA_locator)

timeout (integer)

Return parameters:

errorcode (integer)

session_nonce (structure) (if successful)

proffered_objective (structure) (if successful)

reason (string) (if negotiation declined)

This function opens a negotiation session. The 'objective' parameter must include the requested value, and its loop count should be set to a suitable value by the ASA. If not, the GRASP default will apply.

Note that a given negotiation session may or may not be a dry-run negotiation; the two modes must not be mixed in a single session.

The 'peer' parameter is the target node; it must be an 'ASA_locator' as returned by discover(). If the peer is null, GRASP discovery is performed first.

If the 'errorcode' return parameter is 0, the negotiation has successfully started. There are then two cases:

1. The 'session_nonce' parameter is null. In this case the negotiation has succeeded (the peer has accepted the request). The returned 'proffered_objective' contains the value accepted by the peer.

2. The 'session_nonce' parameter is not null. In this case negotiation must continue. The returned 'proffered_objective' contains the first value proffered by the negotiation peer. Note that this instance of the objective must be used in the subsequent negotiation call because it also contains the current loop count. The 'session_nonce' must be presented in all subsequent negotiation steps.

   This function must be followed by calls to 'negotiate_step' and/or 'negotiate_wait' and/or 'end_negotiate' until the negotiation ends. 'request_negotiate' may then be called again to start a new negotation.

If the 'errorcode' parameter has the value 1 ('declined'), the negotiation has been declined by the peer (M_END and O_DECLINE features of GRASP). The 'reason' string is then available for information and diagnostic use, but it may be a null string. For this and any other error code, an exponential backoff is recommended before any retry.

This should be called in a separate thread if asynchronous operation is required.

Special note for the ACP infrastructure ASA: It is likely that this ASA will need to discover and negotiate with its peers in each of its on-link neighbors. It will therefore need to know not only the link-local IP address but also the physical interface and transport port for connecting to each neighbor. One implementation approach to this is to include these details in the 'session_nonce' data structure, which is opaque to normal ASAs.

- listen_negotiate()

  Input parameters:

    asa_nonce (integer)

    objective (structure)

  Return parameters:

    errorcode (integer)

    session_nonce (structure) (if successful)

    requested_objective (structure) (if successful)

  This function instructs GRASP to listen for negotiation requests for the given 'objective'. It also enables discovery responses for the objective. It will block waiting for an incoming request, so should be called in a separate thread if asynchronous operation is required. Unless there is an unexpected failure, this call only returns after an incoming negotiation request. When it does so, 'requested_objective' contains the first value requested by the negotiation peer. Note that this instance of the objective must be used in the subsequent negotiation call because it also contains the current loop count. The 'session_nonce' must be presented in all subsequent negotiation steps.

This function must be followed by calls to 'negotiate_step' and/or 'negotiate_wait' and/or 'end_negotiate' until the negotiation ends. 'listen_negotiate' may then be called again to await a new negotation.

If an ASA is capable of handling multiple negotiations simultaneously, it may call 'listen_negotiate' simultaneously from multiple threads. The API and GRASP implementation must support re-entrant use of the listening state and the negotiation calls. Simultaneous sessions will be distinguished by the threads themselves, the GRASP Session IDs, and the underlying unicast transport sockets.

- stop_listen_negotiate()

  Input parameters:

    asa_nonce (integer)

    objective (structure)

  Return parameter:

    errorcode (integer)

  Instructs GRASP to stop listening for negotiation requests for the given objective, i.e., cancels 'listen_negotiate'. Of course, it must be called from a different thread.

- negotiate_step()

  Input parameters:

    asa_nonce (integer)

    session_nonce (structure)

    objective (structure)

    timeout (integer)

  Return parameters:

    Exactly as for 'request_negotiate'

  Executes the next negotation step with the peer. The 'objective' parameter contains the next value being proffered by the ASA in this step.

- negotiate_wait()

  Input parameters:

  asa_nonce (integer)

  session_nonce (structure)

  timeout (integer)

  Return parameters:

  errorcode (integer)

  Delay negotiation session by 'timeout' milliseconds.

- end_negotiate()

  Input parameters:

  asa_nonce (integer)

  session_nonce (structure)

  reply (Boolean)

  reason (UTF-8 string)

  Return parameters:

  errorcode (integer)

  End the negotiation session.

  'reply' = True for accept (successful negotiation), False for decline (failed negotiation).

  'reason' = optional string describing reason for decline.

2.2.5. Synchronization and Flooding

- synchronize()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

peer (ASA_locator)

timeout (integer)

Return parameters:

errorcode (integer)

objective (structure) (if successful)

This call requests the synchronized value of the given 'objective'.

Since this is essentially a read operation, any ASA can do it. Therefore the API checks that the ASA is registered but the objective doesn't need to be registered by the calling ASA.

If the objective was already flooded, the flooded value is returned immediately in the 'result' parameter. In this case, the 'source' and 'timeout' are ignored.

Otherwise, synchronization with a discovered ASA is performed. The 'peer' parameter is an 'ASA_locator' as returned by discover(). If 'peer' is null, GRASP discovery is performed first.

This call should be repeated whenever the latest value is needed.

Call in a separate thread if asynchronous operation is required.

Since this is essentially a read operation, any ASA can use it. Therefore GRASP checks that the calling ASA is registered but the objective doesn't need to be registered by the calling ASA.

In the case of failure, an exponential backoff is recommended before retrying.

- listen_synchronize()

Input parameters:

asa_nonce (integer)

objective (structure)

Return parameters:

errorcode (integer)

This instructs GRASP to listen for synchronization requests for the given objective, and to respond with the value given in the 'objective' parameter. It also enables discovery responses for the objective.

This call is non-blocking and may be repeated whenever the value changes.

- stop_listen_synchronize()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

  Return parameters:

  errorcode (integer)

  This call instructs GRASP to stop listening for synchronization requests for the given 'objective', i.e. it cancels a previous listen_synchronize.

- flood()

  Input parameters:

  asa_nonce (integer)

  ttl (integer)

  tagged_objective_list (structure)

  Return parameters:

  errorcode (integer)

  This call instructs GRASP to flood the given synchronization objective(s) and their value(s) and associated locator(s) to all GRASP nodes.

  The 'ttl' parameter is the valid lifetime (time to live) of the flooded data in milliseconds (0 = infinity)

The 'tagged_objective_list' parameter is a list of one or more 'tagged_objective' couplets. The 'locator' parameter that tags each objective is normally null but may be a valid 'ASA_locator'. Infrastructure ASAs needing to flood an {address, protocol, port} 3-tuple with an objective create an ASA_locator object to do so. If the IP address in that locator is the unspecified address ('::') it is replaced by the link-local address of the sending node in each copy of the flood multicast, which will be forced to have a loop count of 1. This feature is for objectives that must be restricted to the local link.

The function checks that the ASA registered each objective.

This call may be repeated whenever any value changes.

- get_flood()

  Input parameters:

    asa_nonce (integer)

    objective (structure)

  Return parameters:

    errorcode (integer)

    tagged_objective_list (structure) (if successful)

  This call instructs GRASP to return the given synchronization objective if it has been flooded and its lifetime has not expired.

  Since this is essentially a read operation, any ASA can do it. Therefore the API checks that the ASA is registered but the objective doesn't need to be registered by the calling ASA.

  The 'tagged_objective_list' parameter is a list of 'tagged_objective' couplets, each one being a copy of the flooded objective and a coresponding locator. Thus if the same objective has been flooded by multiple ASAs, the recipient can distinguish the copies.

  Note that this call is for advanced ASAs. In a simple case, an ASA can simply call synchronize() in order to get a valid flooded objective.

- expire_flood()

  Input parameters:

  asa_nonce (integer)

  tagged_objective (structure)

  Return parameters:

  errorcode (integer)

  This is a call that can only be used after a preceding call to get_flood() by an ASA that is capable of deciding that the flooded value is stale or invalid. Use with care.

  The 'tagged_objective' parameter is the one to be expired.

3. Non-threaded Implementations

If an operating system or language does not provide convenient support for multi-threading, ASAs may need to be written using a polling or 'event loop' structure, whereby a main loop supports multiple GRASP sessions in parallel by repeatedly checking each one for a change of state. To facilitate this, an API implementation may provide alternative versions of all the functions that involve blocking and queueing. In the calls, the error code 2 ("noReply") will be returned by each call instead of blocking, until such time as the event for which it is waiting has been queued. Thus, for example, request_negotiate() would return "noReply" instead of waiting until an incoming request or timeout arrived, and an identical call to request_negotiate() would be repeated in the next cycle of the main loop. In the case of negotiations, the session_nonce parameter is used to distinguish sessions from each other, if necessary.

The calls to which this mechanism applies are:

discover()

request_negotiate()

negotiate_step()

listen_negotiate()

synchronize()

4. Example Logic Flows

TBD

(Until this section is written, some Python examples can be found at <https://www.cs.auckland.ac.nz/~brian/graspy/Briggs.py>, <https://www.cs.auckland.ac.nz/~brian/graspy/Gray.py>, and <https://www.cs.auckland.ac.nz/~brian/graspy/pfxm3.py/>.)

5. Security Considerations

Security issues for the GRASP protocol are discussed in [I-D.ietf-anima-grasp]. Authorization of ASAs is a subject for future study.

The 'asa_nonce' parameter is used in the API as a first line of defence against a malware process attempting to imitate a legitimately registered ASA. The 'session_nonce' parameter is used in the API as a first line of defence against a malware process attempting to hijack a GRASP session.

6. IANA Considerations

This does not need IANA assignment.

7. Acknowledgements

This document was produced using the xml2rfc tool [RFC7749].

Excellent suggestions were made by Michael Richardson.

8. References

8.1. Normative References

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", draft-ietf-anima-grasp-13 (work in progress), June 2017.

8.2. Informative References

[I-D.ietf-anima-autonomic-control-plane]
Behringer, M., Eckert, T., and S. Bjarnason, "An Autonomic Control Plane", draft-ietf-anima-autonomic-control-plane-06 (work in progress), March 2017.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-06 (work in progress), May 2017.

[I-D.ietf-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., Pierre, P., Liu, B., Nobre, J., and J. Strassner, "A Reference Model for Autonomic Networking", draft-ietf-anima-reference-model-03 (work in progress), March 2017.

[I-D.liu-anima-grasp-distribution]
Liu, B. and S. Jiang, "Information Distribution over GRASP", draft-liu-anima-grasp-distribution-04 (work in progress), May 2017.

[RFC7749] Reschke, J., "The "xml2rfc" Version 2 Vocabulary", RFC 7749, DOI 10.17487/RFC7749, February 2016, <http://www.rfc-editor.org/info/rfc7749>.

Appendix A. Error Codes

This Appendix lists the error codes defined so far, with suggested symbolic names and corresponding descriptive strings in English. It is expected that complete API implementations will provide for localisation of these descriptive strings.

```
   ok               0 "OK"
   declined         1 "Declined"
   noReply          2 "No reply"
   unspec           3 "Unspecified error"
   ASAfull          4 "ASA registry full"
   dupASA           5 "Duplicate ASA name"
   noASA            6 "ASA not registered"
   notYourASA       7 "ASA registered but not by you"
   notBoth          8 "Objective cannot support both negotiation
                       and synchronization"
   notDry           9 "Dry-run allowed only with negotiation"
   notOverlap      10 "Overlap not supported by this implementation"
   objFull         11 "Objective registry full"
   objReg          12 "Objective already registered"
   notYourObj      13 "Objective not registered by this ASA"
   notObj          14 "Objective not found"
   notNeg          15 "Objective not negotiable"
   noSecurity      16 "No security"
   noDiscReply     17 "No reply to discovery"
   sockErrNegRq    18 "Socket error sending negotiation request"
   noSession       19 "No session"
   noSocket        20 "No socket"
   loopExhausted   21 "Loop count exhausted"
   sockErrNegStep  22 "Socket error sending negotiation step"
   noPeer          23 "No negotiation peer"
   CBORfail        24 "CBOR decode failure"
   invalidNeg      25 "Invalid Negotiate message"
   invalidEnd      26 "Invalid end message"
   noNegReply      27 "No reply to negotiation step"
   noValidStep     28 "No valid reply to negotiation step"
   sockErrWait     29 "Socket error sending wait message"
   sockErrEnd      30 "Socket error sending end message"
   IDclash         31 "Incoming request Session ID clash"
   notSynch        32 "Not a synchronization objective"
   notFloodDisc    33 "Not flooded and no reply to discovery"
   sockErrSynRq    34 "Socket error sending synch request"
   noListener      35 "No synch listener"
   noSynchReply    36 "No reply to synchronization request"
   noValidSynch    37 "No valid reply to synchronization request"
   invalidLoc      38 "Invalid locator"
```

Appendix B. Change log [RFC Editor: Please remove]

draft-liu-anima-grasp-api-04, 2017-06-30:

Noted that simple nodes might not include the API.

Minor clarifications.

draft-liu-anima-grasp-api-03, 2017-02-13:

Changed error return to integers.

Required all implementations to accept objective values in CBOR.

Added non-blocking alternatives.

draft-liu-anima-grasp-api-02, 2016-12-17:

Updated for draft-ietf-anima-grasp-09

draft-liu-anima-grasp-api-02, 2016-09-30:

Added items for draft-ietf-anima-grasp-07

Editorial corrections

draft-liu-anima-grasp-api-01, 2016-06-24:

Updated for draft-ietf-anima-grasp-05

Editorial corrections

draft-liu-anima-grasp-api-00, 2016-04-04:

Initial version

Authors' Addresses

Brian Carpenter
Department of Computer Science
University of Auckland
PB 92019
Auckland 1142
New Zealand

Email: brian.e.carpenter@gmail.com

Bing Liu (editor)
Huawei Technologies
Q22, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: leo.liubing@huawei.com

Wendong Wang
BUPT University
Beijing University of Posts & Telecom.
No.10 Xitucheng Road
Hai-Dian District, Beijing 100876
P.R. China

Email: wdwang@bupt.edu.cn

Xiangyang Gong
BUPT University
Beijing University of Posts & Telecom.
No.10 Xitucheng Road
Hai-Dian District, Beijing 100876
P.R. China

Email: xygong@bupt.edu.cn