Stream:	Internet Engineering Task Force (IETF)		
RFC:	8990		
Category:	Standards Track		
Published:	May 2021		
ISSN:	2070-1721		
Authors:	C. Bormann	B. Carpenter, Ed.	B. Liu, Ed.
	Universität Bremen TZI	*Univ. of Auckland*	*Huawei Technologies Co., Ltd*

RFC 8990
GeneRic Autonomic Signaling Protocol (GRASP)

Abstract

This document specifies the GeneRic Autonomic Signaling Protocol (GRASP), which enables autonomic nodes and Autonomic Service Agents to dynamically discover peers, to synchronize state with each other, and to negotiate parameter settings with each other. GRASP depends on an external security environment that is described elsewhere. The technical objectives and parameters for specific application scenarios are to be described in separate documents. Appendices briefly discuss requirements for the protocol and existing protocols with comparable features.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8990.

Copyright Notice

Table of Contents

1. Introduction

The success of the Internet has made IP-based networks bigger and more complicated. Large-scale ISP and enterprise networks have become more and more problematic for human-based management. Also, operational costs are growing quickly. Consequently, there are increased requirements for autonomic behavior in the networks. General aspects of Autonomic Networks are discussed in [RFC7575] and [RFC7576].

One approach is to largely decentralize the logic of network management by migrating it into network elements. A reference model for Autonomic Networking on this basis is given in [RFC8993]. The reader should consult this document to understand how various autonomic components fit together. In order to achieve autonomy, devices that embody Autonomic Service Agents (ASAs, [RFC7575]) have specific signaling requirements. In particular, they need to discover each other, to synchronize state with each other, and to negotiate parameters and resources directly with each other. There is no limitation on the types of parameters and resources concerned, which can include very basic information needed for addressing and routing, as well as anything else that might be configured in a conventional non-autonomic network. The atomic unit of discovery, synchronization, or negotiation is referred to as a technical objective, i.e., a configurable parameter or set of parameters (defined more precisely in Section 2.1).

Negotiation is an iterative process, requiring multiple message exchanges forming a closed loop between the negotiating entities. In fact, these entities are ASAs, normally but not necessarily in different network devices. State synchronization, when needed, can be regarded as a special case of negotiation without iteration. Both negotiation and synchronization must logically follow discovery. More details of the requirements are found in Appendix B. Section 2.3 describes a behavior model for a protocol intended to support discovery, synchronization, and negotiation. The design of GeneRic Autonomic Signaling Protocol (GRASP) in Section 2 is based on this behavior model. The relevant capabilities of various existing protocols are reviewed in Appendix C.

The proposed discovery mechanism is oriented towards synchronization and negotiation objectives. It is based on a neighbor discovery process on the local link, but it also supports diversion to peers on other links. There is no assumption of any particular form of network topology. When a device starts up with no preconfiguration, it has no knowledge of the topology. The protocol itself is capable of being used in a small and/or flat network structure such as a

small office or home network as well as in a large, professionally managed network. Therefore, the discovery mechanism needs to be able to allow a device to bootstrap itself without making any prior assumptions about network structure.

Because GRASP can be used as part of a decision process among distributed devices or between networks, it must run in a secure and strongly authenticated environment.

In realistic deployments, not all devices will support GRASP. Therefore, some Autonomic Service Agents will directly manage a group of non-autonomic nodes, and other non-autonomic nodes will be managed traditionally. Such mixed scenarios are not discussed in this specification.

2. Protocol Overview

2.1. Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses terminology defined in [RFC7575].

The following additional terms are used throughout this document:

Discovery:
: A process by which an ASA discovers peers according to a specific discovery objective. The discovery results may be different according to the different discovery objectives. The discovered peers may later be used as negotiation counterparts or as sources of synchronization data.

Negotiation:
: A process by which two ASAs interact iteratively to agree on parameter settings that best satisfy the objectives of both ASAs.

State Synchronization:
: A process by which ASAs interact to receive the current state of parameter values stored in other ASAs. This is a special case of negotiation in which information is sent, but the ASAs do not request their peers to change parameter settings. All other definitions apply to both negotiation and synchronization.

Technical Objective (usually abbreviated as Objective):
: A technical objective is a data structure whose main contents are a name and a value. The value consists of a single configurable parameter or a set of parameters of some kind. The exact format of an objective is defined in Section 2.10.1. An objective occurs in three contexts: discovery, negotiation, and synchronization. Normally, a given objective will not occur in negotiation and synchronization contexts simultaneously.

  One ASA may support multiple independent objectives.

  The parameter(s) in the value of a given objective apply to a specific service or function or action. They may in principle be anything that can be set to a specific logical, numerical, or string value, or a more complex data structure, by a network node. Each node is expected to contain one or more ASAs which may each manage subsidiary non-autonomic nodes.

  Discovery Objective: an objective in the process of discovery. Its value may be undefined.

  Synchronization Objective: an objective whose specific technical content needs to be synchronized among two or more ASAs. Thus, each ASA will maintain its own copy of the objective.

  Negotiation Objective: an objective whose specific technical content needs to be decided in coordination with another ASA. Again, each ASA will maintain its own copy of the objective.

  A detailed discussion of objectives, including their format, is found in Section 2.10.

Discovery Initiator:
: An ASA that starts discovery by sending a Discovery message referring to a specific discovery objective.

Discovery Responder:
: A peer that either contains an ASA supporting the discovery objective indicated by the discovery initiator or caches the locator(s) of the ASA(s) supporting the objective. It sends a Discovery Response, as described later.

Synchronization Initiator:
: An ASA that starts synchronization by sending a request message referring to a specific synchronization objective.

Synchronization Responder:
: A peer ASA that responds with the value of a synchronization objective.

Negotiation Initiator:
: An ASA that starts negotiation by sending a request message referring to a specific negotiation objective.

Negotiation Counterpart:
: A peer with which the negotiation initiator negotiates a specific negotiation objective.

GRASP Instance:
: This refers to an instantiation of a GRASP protocol engine, likely including multiple threads or processes as well as dynamic data structures such as a discovery cache, running in a given security environment on a single device.

GRASP Core:
: This refers to the code and shared data structures of a GRASP instance, which will communicate with individual ASAs via a suitable Application Programming Interface (API).

Interface or GRASP Interface:
: Unless otherwise stated, this refers to a network interface, which might be physical or virtual, that a specific instance of GRASP is currently using. A device might have other interfaces that are not used by GRASP and which are outside the scope of the Autonomic Network.

2.2. High-Level Deployment Model

A GRASP implementation will be part of the Autonomic Networking Infrastructure (ANI) in an autonomic node, which must also provide an appropriate security environment. In accordance with [RFC8993], this **SHOULD** be the Autonomic Control Plane (ACP) [RFC8994]. As a result, all autonomic nodes in the ACP are able to trust each other. It is expected that GRASP will access the ACP by using a typical socket programming interface, and the ACP will make available only network interfaces within the Autonomic Network. If there is no ACP, the considerations described in Section 2.5.1 apply.

There will also be one or more Autonomic Service Agents (ASAs). In the minimal case of a single-purpose device, these components might be fully integrated with GRASP and the ACP. A more common model is expected to be a multipurpose device capable of containing several ASAs, such as a router or large switch. In this case it is expected that the ACP, GRASP and the ASAs will be implemented as separate processes, which are able to support asynchronous and simultaneous operations, for example by multithreading.

In some scenarios, a limited negotiation model might be deployed based on a limited trust relationship such as that between two administrative domains. ASAs might then exchange limited information and negotiate some particular configurations.

GRASP is explicitly designed to operate within a single addressing realm. Its discovery and flooding mechanisms do not support autonomic operations that cross any form of address translator or upper-layer proxy.

A suitable Application Programming Interface (API) will be needed between GRASP and the ASAs. In some implementations, ASAs would run in user space with a GRASP library providing the API, and this library would in turn communicate via system calls with core GRASP functions. Details of the API are out of scope for the present document. For further details of possible deployment models, see [RFC8993].

An instance of GRASP must be aware of the network interfaces it will use, and of the appropriate global-scope and link-local addresses. In the presence of the ACP, such information will be available from the adjacency table discussed in [RFC8993]. In other cases, GRASP must determine

such information for itself. Details depend on the device and operating system. In the rest of this document, the terms 'interfaces' or 'GRASP interfaces' refers only to the set of network interfaces that a specific instance of GRASP is currently using.

Because GRASP needs to work with very high reliability, especially during bootstrapping and during fault conditions, it is essential that every implementation continues to operate in adverse conditions. For example, discovery failures, or any kind of socket exception at any time, must not cause irrecoverable failures in GRASP itself, and must return suitable error codes through the API so that ASAs can also recover.

GRASP must not depend upon nonvolatile data storage. All runtime error conditions, and events such as address renumbering, network interface failures, and CPU sleep/wake cycles, must be handled in such a way that GRASP will still operate correctly and securely afterwards (Section 2.5.1).

An autonomic node will normally run a single instance of GRASP, which is used by multiple ASAs. Possible exceptions are mentioned below.

2.3. High-Level Design

This section describes the behavior model and general design of GRASP, supporting discovery, synchronization, and negotiation, to act as a platform for different technical objectives.

A generic platform:
: The protocol design is generic and independent of the synchronization or negotiation contents. The technical contents will vary according to the various technical objectives and the different pairs of counterparts.

Multiple instances:
: Normally, a single main instance of the GRASP protocol engine will exist in an autonomic node, and each ASA will run as an independent asynchronous process. However, scenarios where multiple instances of GRASP run in a single node, perhaps with different security properties, are possible (Section 2.5.2). In this case, each instance **MUST** listen independently for GRASP link-local multicasts, and all instances **MUST** be woken by each such multicast in order for discovery and flooding to work correctly.

Security infrastructure:
: As noted above, the protocol itself has no built-in security functionality and relies on a separate secure infrastructure.

Discovery, synchronization, and negotiation are designed together:
: The discovery method and the synchronization and negotiation methods are designed in the same way and can be combined when this is useful, allowing a rapid mode of operation described in Section 2.5.4. These processes can also be performed independently when appropriate.

  Thus, for some objectives, especially those concerned with application-layer services, another discovery mechanism such as DNS-based Service Discovery [RFC7558] **MAY** be used. The choice is left to the designers of individual ASAs.

A uniform pattern for technical objectives:
: The synchronization and negotiation objectives are defined according to a uniform pattern. The values that they contain could be carried either in a simple binary format or in a complex object format. The basic protocol design uses the Concise Binary Object Representation (CBOR) [RFC8949], which is readily extensible for unknown, future requirements.

A flexible model for synchronization:
: GRASP supports synchronization between two nodes, which could be used repeatedly to perform synchronization among a small number of nodes. It also supports an unsolicited flooding mode when large groups of nodes, possibly including all autonomic nodes, need data for the same technical objective.

  There may be some network parameters for which a more traditional flooding mechanism such as the Distributed Node Consensus Protocol (DNCP) [RFC7787] is considered more appropriate. GRASP can coexist with DNCP.

A simple initiator/responder model for negotiation:
: Multiparty negotiations are very complicated to model and cannot readily be guaranteed to converge. GRASP uses a simple bilateral model and can support multiparty negotiations by indirect steps.

Organizing of synchronization or negotiation content:
: The technical content transmitted by GRASP will be organized according to the relevant function or service. The objectives for different functions or services are kept separate because they may be negotiated or synchronized with different counterparts or have different response times. Thus a normal arrangement is a single ASA managing a small set of closely related objectives, with a version of that ASA in each relevant autonomic node. Further discussion of this aspect is out of scope for the current document.

Requests and responses in negotiation procedures:
: The initiator can negotiate a specific negotiation objective with relevant counterpart ASAs. It can request relevant information from a counterpart so that it can coordinate its local configuration. It can request the counterpart to make a matching configuration. It can request simulation or forecast results by sending some dry-run conditions.

  Beyond the traditional yes/no answer, the responder can reply with a suggested alternative value for the objective concerned. This would start a bidirectional negotiation ending in a compromise between the two ASAs.

Convergence of negotiation procedures:
: To enable convergence when a responder suggests a new value or condition in a negotiation step reply, it should be as close as possible to the original request or previous suggestion. The suggested value of later negotiation steps should be chosen between the suggested values from the previous two steps. GRASP provides mechanisms to guarantee convergence (or failure) in a small number of steps, namely a timeout and a maximum number of iterations.

Extensibility:
: GRASP intentionally does not have a version number, and it can be extended by adding new message types and options. The Invalid message (M_INVALID) will be used to signal that an implementation does not recognize a message or option sent by another implementation. In normal use, new semantics will be added by defining new synchronization or negotiation objectives.

2.4. Quick Operating Overview

An instance of GRASP is expected to run as a separate core module, providing an API (such as [RFC8991]) to interface to various ASAs. These ASAs may operate without special privilege, unless they need it for other reasons (such as configuring IP addresses or manipulating routing tables).

The GRASP mechanisms used by the ASA are built around GRASP objectives defined as data structures containing administrative information such as the objective's unique name and its current value. The format and size of the value is not restricted by the protocol, except that it must be possible to serialize it for transmission in CBOR, which is no restriction at all in practice.

GRASP provides the following mechanisms:

- A discovery mechanism (M_DISCOVERY, M_RESPONSE) by which an ASA can discover other ASAs supporting a given objective.
- A negotiation request mechanism (M_REQ_NEG) by which an ASA can start negotiation of an objective with a counterpart ASA. Once a negotiation has started, the process is symmetrical, and there is a negotiation step message (M_NEGOTIATE) for each ASA to use in turn. Two other functions support negotiating steps (M_WAIT, M_END).
- A synchronization mechanism (M_REQ_SYN) by which an ASA can request the current value of an objective from a counterpart ASA. With this, there is a corresponding response function (M_SYNCH) for an ASA that wishes to respond to synchronization requests.
- A flood mechanism (M_FLOOD) by which an ASA can cause the current value of an objective to be flooded throughout the Autonomic Network so that any ASA can receive it. One application of this is to act as an announcement, avoiding the need for discovery of a widely applicable objective.

Some example messages and simple message flows are provided in Appendix A.

2.5. GRASP Basic Properties and Mechanisms

2.5.1. Required External Security Mechanism

GRASP does not specify transport security because it is meant to be adapted to different environments. Every solution adopting GRASP **MUST** specify a security and transport substrate used by GRASP in that solution.

The substrate **MUST** enforce sending and receiving GRASP messages only between members of a mutually trusted group running GRASP. Each group member is an instance of GRASP. The group members are nodes of a connected graph. The group and graph are created by the security and transport substrate and are called the GRASP domain. The substrate must support unicast messages between any group members and (link-local) multicast messages between adjacent group members. It must deny messages between group members and non-group members. With this model, security is provided by enforcing group membership, but any member of the trusted group can attack the entire network until revoked.

Substrates **MUST** use cryptographic member authentication and message integrity for GRASP messages. This can be end to end or hop by hop across the domain. The security and transport substrate **MUST** provide mechanisms to remove untrusted members from the group.

If the substrate does not mandate and enforce GRASP message encryption, then any service using GRASP in such a solution **MUST** provide protection and encryption for message elements whose exposure could constitute an attack vector.

The security and transport substrate for GRASP in the ANI is the ACP. Unless otherwise noted, we assume this security and transport substrate in the remainder of this document. The ACP does mandate the use of encryption; therefore, GRASP in the ANI can rely on GRASP messages being encrypted. The GRASP domain is the ACP: all nodes in an autonomic domain connected by encrypted virtual links formed by the ACP. The ACP uses hop-by-hop security (authentication and encryption) of messages. Removal of nodes relies on standard PKI certificate revocation or expiry of sufficiently short-lived certificates. Refer to [RFC8994] for more details.

As mentioned in Section 2.3, some GRASP operations might be performed across an administrative domain boundary by mutual agreement, without the benefit of an ACP. Such operations **MUST** be confined to a separate instance of GRASP with its own copy of all GRASP data structures running across a separate GRASP domain with a security and transport substrate. In the most simple case, each point-to-point interdomain GRASP peering could be a separate domain, and the security and transport substrate could be built using transport or network-layer security protocols. This is subject to future specifications.

An exception to the requirements for the security and transport substrate exists for highly constrained subsets of GRASP meant to support the establishment of a security and transport substrate, described in the following section.

2.5.2. Discovery Unsolicited Link-Local (DULL) GRASP

Some services may need to use insecure GRASP discovery, response, and flood messages without being able to use preexisting security associations, for example, as part of discovery for establishing security associations such as a security substrate for GRASP.

Such operations being intrinsically insecure, they need to be confined to link-local use to minimize the risk of malicious actions. Possible examples include discovery of candidate ACP neighbors [RFC8994], discovery of bootstrap proxies [RFC8995], or perhaps initialization services in networks using GRASP without being fully autonomic (e.g., no ACP). Such usage **MUST** be limited to link-local operations on a single interface and **MUST** be confined to a separate insecure instance of GRASP with its own copy of all GRASP data structures. This instance is nicknamed DULL -- Discovery Unsolicited Link-Local.

The detailed rules for the DULL instance of GRASP are as follows:

- An initiator **MAY** send Discovery or Flood Synchronization link-local multicast messages that **MUST** have a loop count of 1, to prevent off-link operations. Other unsolicited GRASP message types **MUST NOT** be sent.
- A responder **MUST** silently discard any message whose loop count is not 1.
- A responder **MUST** silently discard any message referring to a GRASP objective that is not directly part of a service that requires this insecure mode.
- A responder **MUST NOT** relay any multicast messages.
- A Discovery Response **MUST** indicate a link-local address.
- A Discovery Response **MUST NOT** include a Divert option.
- A node **MUST** silently discard any message whose source address is not link-local.

To minimize traffic possibly observed by third parties, GRASP traffic **SHOULD** be minimized by using only Flood Synchronization to announce objectives and their associated locators, rather than by using Discovery and Discovery Response messages. Further details are out of scope for this document.

2.5.3. Transport Layer Usage

All GRASP messages, after they are serialized as a CBOR byte string, are transmitted as such directly over the transport protocol in use. The transport protocol(s) for a GRASP domain are specified by the security and transport substrate as introduced in Section 2.5.1.

GRASP discovery and flooding messages are designed for GRASP domain-wide flooding through hop-by-hop link-local multicast forwarding between adjacent GRASP nodes. The GRASP security and transport substrate needs to specify how these link-local multicasts are transported. This can be unreliable transport (UDP) but it **SHOULD** be reliable transport (e.g., TCP).

If the substrate specifies an unreliable transport such as UDP for discovery and flooding messages, then it **MUST NOT** use IP fragmentation because of its loss characteristic, especially in multi-hop flooding. GRASP **MUST** then enforce at the user API level a limit to the size of discovery

and flooding messages, so that no fragmentation can occur. For IPv6 transport, this means that the size of those messages' IPv6 packets must be at most 1280 bytes (unless there is a known larger minimum link MTU across the whole GRASP domain).

All other GRASP messages are unicast between group members of the GRASP domain. These **MUST** use a reliable transport protocol because GRASP itself does not provide for error detection, retransmission, or flow control. Unless otherwise specified by the security and transport substrate, TCP **MUST** be used.

The security and transport substrate for GRASP in the ANI is the ACP. Unless otherwise noted, we assume this security and transport substrate in the remainder of this document when describing GRASP's message transport. In the ACP, TCP is used for GRASP unicast messages. GRASP discovery and flooding messages also use TCP: these link-local messages are forwarded by replicating them to all adjacent GRASP nodes on the link via TCP connections to those adjacent GRASP nodes. Because of this, GRASP in the ANI has no limitations on the size of discovery and flooding messages with respect to fragmentation issues. While the ACP is being built using a DULL instance of GRASP, native UDP multicast is used to discover ACP/GRASP neighbors on links.

For link-local UDP multicast, GRASP listens to the well-known GRASP Listen Port (Section 2.6). Transport connections for discovery and flooding on relay nodes must terminate in GRASP instances (e.g., GRASP ASAs) so that link-local multicast, hop-by-hop flooding of M_DISCOVERY and M_FLOOD messages and hop-by-hop forwarding of M_RESPONSE responses and caching of those responses along the path work correctly.

Unicast transport connections used for synchronization and negotiation can terminate directly in ASAs that implement objectives; therefore, this traffic does not need to pass through GRASP instances. For this, the ASA listens on its own dynamically assigned ports, which are communicated to its peers during discovery. Alternatively, the GRASP instance can also terminate the unicast transport connections and pass the traffic from/to the ASA if that is preferable in some implementations (e.g., to better decouple ASAs from network connections).

2.5.4. Discovery Mechanism and Procedures

2.5.4.1. Separated Discovery and Negotiation Mechanisms

Although discovery and negotiation or synchronization are defined together in GRASP, they are separate mechanisms. The discovery process could run independently from the negotiation or synchronization process. Upon receiving a Discovery message (Section 2.8.4), the recipient node should return a Discovery Response message in which it either indicates itself as a discovery responder or diverts the initiator towards another more suitable ASA. However, this response may be delayed if the recipient needs to relay the Discovery message onward, as described in Section 2.5.4.4.

The discovery action (M_DISCOVERY) will normally be followed by a negotiation (M_REQ_NEG) or synchronization (M_REQ_SYN) action. The discovery results could be utilized by the negotiation protocol to decide which ASA the initiator will negotiate with.

The initiator of a discovery action for a given objective need not be capable of responding to that objective as a negotiation counterpart, as a synchronization responder, or as source for flooding. For example, an ASA might perform discovery even if it only wishes to act as a synchronization initiator or negotiation initiator. Such an ASA does not itself need to respond to Discovery messages.

It is also entirely possible to use GRASP discovery without any subsequent negotiation or synchronization action. In this case, the discovered objective is simply used as a name during the discovery process, and any subsequent operations between the peers are outside the scope of GRASP.

2.5.4.2. Discovery Overview

A complete discovery process will start with a multicast Discovery message (M_DISCOVERY) on the local link. On-link neighbors supporting the discovery objective will respond directly with Discovery Response (M_RESPONSE) messages. A neighbor with multiple interfaces may respond with a cached Discovery Response. If it has no cached response, it will relay the Discovery message on its other GRASP interfaces. If a node receiving the relayed Discovery message supports the discovery objective, it will respond to the relayed Discovery message. If it has a cached response, it will respond with that. If not, it will repeat the discovery process, which thereby becomes iterative. The loop count and timeout will ensure that the process ends. Further details are given in Section 2.5.4.4.

A Discovery message **MAY** be sent unicast to a peer node, which **SHOULD** then proceed exactly as if the message had been multicast, except that when TCP is used, the response will be on the same socket as the query. However, this mode does not guarantee successful discovery in the general case.

2.5.4.3. Discovery Procedures

Discovery starts as an on-link operation. The Divert option can tell the discovery initiator to contact an off-link ASA for that discovery objective. If the security and transport substrate of the GRASP domain (see Section 2.5.3) uses UDP link-local multicast, then the discovery initiator sends these to the ALL_GRASP_NEIGHBORS link-local multicast address (Section 2.6), and all GRASP nodes need to listen to this address to act as discovery responders. Because this port is unique in a device, this is a function of the GRASP instance and not of an individual ASA. As a result, each ASA will need to register the objectives that it supports with the local GRASP instance.

If an ASA in a neighbor device supports the requested discovery objective, the device **SHOULD** respond to the link-local multicast with a unicast Discovery Response message (Section 2.8.5) with locator option(s) (Section 2.9.5) unless it is temporarily unavailable. Otherwise, if the neighbor has cached information about an ASA that supports the requested discovery objective (usually because it discovered the same objective before), it **SHOULD** respond with a Discovery Response message with a Divert option pointing to the appropriate discovery responder. However, it **SHOULD NOT** respond with a cached response on an interface if it learned that information from the same interface because the peer in question will answer directly if still operational.

If a device has no information about the requested discovery objective and is not acting as a discovery relay (see Section 2.5.4.4), it **MUST** silently discard the Discovery message.

The discovery initiator **MUST** set a reasonable timeout on the discovery process. A suggested value is 100 milliseconds multiplied by the loop count embedded in the objective.

If no Discovery Response is received within the timeout, the Discovery message **MAY** be repeated with a newly generated Session ID (Section 2.7). An exponential backoff **SHOULD** be used for subsequent repetitions to limit the load during busy periods. The details of the backoff algorithm will depend on the use case for the objective concerned but **MUST** be consistent with the recommendations in [RFC8085] for low data-volume multicast. Frequent repetition might be symptomatic of a denial-of-service attack.

After a GRASP device successfully discovers a locator for a discovery responder supporting a specific objective, it **SHOULD** cache this information, including the interface index [RFC3493] via which it was discovered. This cache record **MAY** be used for future negotiation or synchronization, and the locator **SHOULD** be passed on when appropriate as a Divert option to another discovery initiator.

The cache mechanism **MUST** include a lifetime for each entry. The lifetime is derived from a time-to-live (ttl) parameter in each Discovery Response message. Cached entries **MUST** be ignored or deleted after their lifetime expires. In some environments, unplanned address renumbering might occur. In such cases, the lifetime **SHOULD** be short compared to the typical address lifetime. The discovery mechanism needs to track the node's current address to ensure that Discovery Responses always indicate the correct address.

If multiple discovery responders are found for the same objective, they **SHOULD** all be cached unless this creates a resource shortage. The method of choosing between multiple responders is an implementation choice. This choice **MUST** be available to each ASA, but the GRASP implementation **SHOULD** provide a default choice.

Because discovery responders will be cached in a finite cache, they might be deleted at any time. In this case, discovery will need to be repeated. If an ASA exits for any reason, its locator might still be cached for some time, and attempts to connect to it will fail. ASAs need to be robust in these circumstances.

2.5.4.4. Discovery Relaying

A GRASP instance with multiple link-layer interfaces (typically running in a router) **MUST** support discovery on all GRASP interfaces. We refer to this as a 'relaying instance'.

DULL instances (Section 2.5.2) are always single-interface instances and therefore **MUST NOT** perform discovery relaying.

If a relaying instance receives a Discovery message on a given interface for a specific objective that it does not support and for which it has not previously cached a discovery responder, it **MUST** relay the query by reissuing a new Discovery message as a link-local multicast on its other GRASP interfaces.

The relayed Discovery message **MUST** have the same Session ID and 'initiator' field as the incoming message (see Section 2.8.4). The IP address in the 'initiator' field is only used to disambiguate the Session ID and is never used to address Response packets. Response packets are sent back to the relaying instance, not the original initiator.

The M_DISCOVERY message does not encode the transport address of the originator or relay. Response packets must therefore be sent to the transport-layer address of the connection on which the M_DISCOVERY message was received. If the M_DISCOVERY was relayed via a reliable hop-by-hop transport connection, the response is simply sent back via the same connection.

If the M_DISCOVERY was relayed via link-local (e.g., UDP) multicast, the response is sent back via a reliable hop-by-hop transport connection with the same port number as the source port of the link-local multicast. Therefore, if link-local multicast is used and M_RESPONSE messages are required (which is the case in almost all GRASP instances except for the limited use of DULL instances in the ANI), GRASP needs to be able to bind to one port number on UDP from which to originate the link-local multicast M_DISCOVERY messages and the same port number on the reliable hop-by-hop transport (e.g., TCP by default) to be able to respond to transport connections from responders that want to send M_RESPONSE messages back. Note that this port does not need to be the GRASP_LISTEN_PORT.

The relaying instance **MUST** decrement the loop count within the objective, and **MUST NOT** relay the Discovery message if the result is zero. Also, it **MUST** limit the total rate at which it relays Discovery messages to a reasonable value in order to mitigate possible denial-of-service attacks. For example, the rate limit could be set to a small multiple of the observed rate of Discovery messages during normal operation. The relaying instance **MUST** cache the Session ID value and initiator address of each relayed Discovery message until any Discovery Responses have arrived or the discovery process has timed out. To prevent loops, it **MUST NOT** relay a Discovery message that carries a given cached Session ID and initiator address more than once. These precautions avoid discovery loops and mitigate potential overload.

Since the relay device is unaware of the timeout set by the original initiator, it **SHOULD** set a suitable timeout for the relayed Discovery message. A suggested value is 100 milliseconds multiplied by the remaining loop count.

The discovery results received by the relaying instance **MUST** in turn be sent as a Discovery Response message to the Discovery message that caused the relay action.

2.5.4.5. Rapid Mode (Discovery with Negotiation or Synchronization)

A Discovery message **MAY** include an objective option. This allows a rapid mode of negotiation (Section 2.5.5.1) or synchronization (Section 2.5.6.3). Rapid mode is currently limited to a single objective for simplicity of design and implementation. A possible future extension is to allow multiple objectives in rapid mode for greater efficiency.

2.5.5. Negotiation Procedures

A negotiation initiator opens a transport connection to a counterpart ASA using the address, protocol, and port obtained during discovery. It then sends a negotiation request (using M_REQ_NEG) to the counterpart, including a specific negotiation objective. It may request the negotiation counterpart to make a specific configuration. Alternatively, it may request a certain simulation or forecast result by sending a dry-run configuration. The details, including the distinction between a dry run and a live configuration change, will be defined separately for each type of negotiation objective. Any state associated with a dry-run operation, such as temporarily reserving a resource for subsequent use in a live run, is entirely a matter for the designer of the ASA concerned.

Each negotiation session as a whole is subject to a timeout (default GRASP_DEF_TIMEOUT milliseconds, Section 2.6), initialized when the request is sent (see Section 2.8.6). If no reply message of any kind is received within the timeout, the negotiation request **MAY** be repeated with a newly generated Session ID (Section 2.7). An exponential backoff **SHOULD** be used for subsequent repetitions. The details of the backoff algorithm will depend on the use case for the objective concerned.

If the counterpart can immediately apply the requested configuration, it will give an immediate positive (O_ACCEPT) answer using the Negotiation End (M_END) message. This will end the negotiation phase immediately. Otherwise, it will negotiate (using M_NEGOTIATE). It will reply with a proposed alternative configuration that it can apply (typically, a configuration that uses fewer resources than requested by the negotiation initiator). This will start a bidirectional negotiation using the Negotiate (M_NEGOTIATE) message to reach a compromise between the two ASAs.

The negotiation procedure is ended when one of the negotiation peers sends a Negotiation End (M_END) message, which contains an Accept (O_ACCEPT) or Decline (O_DECLINE) option and does not need a response from the negotiation peer. Negotiation may also end in failure (equivalent to a decline) if a timeout is exceeded or a loop count is exceeded. When the procedure ends for whatever reason, the transport connection **SHOULD** be closed. A transport session failure is treated as a negotiation failure.

A negotiation procedure concerns one objective and one counterpart. Both the initiator and the counterpart may take part in simultaneous negotiations with various other ASAs or in simultaneous negotiations about different objectives. Thus, GRASP is expected to be used in a multithreaded mode or its logical equivalent. Certain negotiation objectives may have restrictions on multithreading, for example to avoid over-allocating resources.

Some configuration actions, for example, wavelength switching in optical networks, might take considerable time to execute. The ASA concerned needs to allow for this by design, but GRASP does allow for a peer to insert latency in a negotiation process if necessary (Section 2.8.9, M_WAIT).

2.5.5.1. Rapid Mode (Discovery/Negotiation Linkage)

A Discovery message **MAY** include a Negotiation Objective option. In this case, it is as if the initiator sent the sequence M_DISCOVERY immediately followed by M_REQ_NEG. This has implications for the construction of the GRASP core, as it must carefully pass the contents of the Negotiation Objective option to the ASA so that it may evaluate the objective directly. When a Negotiation Objective option is present, the ASA replies with an M_NEGOTIATE message (or M_END with O_ACCEPT if it is immediately satisfied with the proposal) rather than with an M_RESPONSE. However, if the recipient node does not support rapid mode, discovery will continue normally.

It is possible that a Discovery Response will arrive from a responder that does not support rapid mode before such a Negotiation message arrives. In this case, rapid mode will not occur.

This rapid mode could reduce the interactions between nodes so that a higher efficiency could be achieved. However, a network in which some nodes support rapid mode and others do not will have complex timing-dependent behaviors. Therefore, the rapid negotiation function **SHOULD** be disabled by default.

2.5.6. Synchronization and Flooding Procedures

2.5.6.1. Unicast Synchronization

A synchronization initiator opens a transport connection to a counterpart ASA using the address, protocol, and port obtained during discovery. It then sends a Request Synchronization message (M_REQ_SYN, Section 2.8.6) to the counterpart, including a specific synchronization objective. The counterpart responds with a Synchronization message (M_SYNCH, Section 2.8.10) containing the current value of the requested synchronization objective. No further messages are needed, and the transport connection **SHOULD** be closed. A transport session failure is treated as a synchronization failure.

If no reply message of any kind is received within a given timeout (default GRASP_DEF_TIMEOUT milliseconds, Section 2.6), the synchronization request **MAY** be repeated with a newly generated Session ID (Section 2.7). An exponential backoff **SHOULD** be used for subsequent repetitions. The details of the backoff algorithm will depend on the use case for the objective concerned.

2.5.6.2. Flooding

In the case just described, the message exchange is unicast and concerns only one synchronization objective. For large groups of nodes requiring the same data, synchronization flooding is available. For this, a flooding initiator **MAY** send an unsolicited Flood Synchronization message (Section 2.8.11) containing one or more Synchronization Objective option(s), if and only if the specification of those objectives permits it. This is sent as a multicast message to the ALL_GRASP_NEIGHBORS multicast address (Section 2.6).

Receiving flood multicasts is a function of the GRASP core, as in the case of discovery multicasts (Section 2.5.4.3).

To ensure that flooding does not result in a loop, the originator of the Flood Synchronization message **MUST** set the loop count in the objectives to a suitable value (the default is GRASP_DEF_LOOPCT). Also, a suitable mechanism is needed to avoid excessive multicast traffic. This mechanism **MUST** be defined as part of the specification of the synchronization objective(s) concerned. It might be a simple rate limit or a more complex mechanism such as the Trickle algorithm [RFC6206].

A GRASP device with multiple link-layer interfaces (typically a router) **MUST** support synchronization flooding on all GRASP interfaces. If it receives a multicast Flood Synchronization message on a given interface, it **MUST** relay it by reissuing a Flood Synchronization message as a link-local multicast on its other GRASP interfaces. The relayed message **MUST** have the same Session ID as the incoming message and **MUST** be tagged with the IP address of its original initiator.

Link-layer flooding is supported by GRASP by setting the loop count to 1 and sending with a link-local source address. Floods with link-local source addresses and a loop count other than 1 are invalid, and such messages **MUST** be discarded.

The relaying device **MUST** decrement the loop count within the first objective and **MUST NOT** relay the Flood Synchronization message if the result is zero. Also, it **MUST** limit the total rate at which it relays Flood Synchronization messages to a reasonable value, in order to mitigate possible denial-of-service attacks. For example, the rate limit could be set to a small multiple of the observed rate of flood messages during normal operation. The relaying device **MUST** cache the Session ID value and initiator address of each relayed Flood Synchronization message for a time not less than twice GRASP_DEF_TIMEOUT milliseconds. To prevent loops, it **MUST NOT** relay a Flood Synchronization message that carries a given cached Session ID and initiator address more than once. These precautions avoid synchronization loops and mitigate potential overload.

Note that this mechanism is unreliable in the case of sleeping nodes, or new nodes that join the network, or nodes that rejoin the network after a fault. An ASA that initiates a flood **SHOULD** repeat the flood at a suitable frequency, which **MUST** be consistent with the recommendations in [RFC8085] for low data-volume multicast. The ASA **SHOULD** also act as a synchronization responder for the objective(s) concerned. Thus nodes that require an objective subject to flooding can either wait for the next flood or request unicast synchronization for that objective.

The multicast messages for synchronization flooding are subject to the security rules in Section 2.5.1. In practice, this means that they **MUST NOT** be transmitted and **MUST** be ignored on receipt unless there is an operational ACP or equivalent strong security in place. However, because of the security weakness of link-local multicast (Section 3), synchronization objectives that are flooded **SHOULD NOT** contain unencrypted private information and **SHOULD** be validated by the recipient ASA.

2.5.6.3. Rapid Mode (Discovery/Synchronization Linkage)

A Discovery message **MAY** include a Synchronization Objective option. In this case, the Discovery message also acts as a Request Synchronization message to indicate to the discovery responder that it could directly reply to the discovery initiator with a Synchronization message (Section

2.8.10) with synchronization data for rapid processing, if the discovery target supports the corresponding synchronization objective. The design implications are similar to those discussed in Section 2.5.5.1.

It is possible that a Discovery Response will arrive from a responder that does not support rapid mode before such a Synchronization message arrives. In this case, rapid mode will not occur.

This rapid mode could reduce the interactions between nodes so that a higher efficiency could be achieved. However, a network in which some nodes support rapid mode and others do not will have complex timing-dependent behaviors. Therefore, the rapid synchronization function **SHOULD** be configured off by default and **MAY** be configured on or off by Intent.

2.6. GRASP Constants

ALL_GRASP_NEIGHBORS
: A link-local scope multicast address used by a GRASP-enabled device to discover GRASP-enabled neighbor (i.e., on-link) devices. All devices that support GRASP are members of this multicast group.

  - IPv6 multicast address: ff02::13
  - IPv4 multicast address: 224.0.0.119

GRASP_LISTEN_PORT (7017)
: A well-known UDP user port that every GRASP-enabled network device **MUST** listen to for link-local multicasts when UDP is used for M_DISCOVERY or M_FLOOD messages in the GRASP instance. This user port **MAY** also be used to listen for TCP or UDP unicast messages in a simple implementation of GRASP (Section 2.5.3).

GRASP_DEF_TIMEOUT (60000 milliseconds)
: The default timeout used to determine that an operation has failed to complete.

GRASP_DEF_LOOPCT (6)
: The default loop count used to determine that a negotiation has failed to complete and to avoid looping messages.

GRASP_DEF_MAX_SIZE (2048)
: The default maximum message size in bytes.

2.7. Session Identifier (Session ID)

This is an up to 32-bit opaque value used to distinguish multiple sessions between the same two devices. A new Session ID **MUST** be generated by the initiator for every new Discovery, Flood Synchronization, or Request message. All responses and follow-up messages in the same discovery, synchronization, or negotiation procedure **MUST** carry the same Session ID.

The Session ID **SHOULD** have a very low collision rate locally. It **MUST** be generated by a pseudorandom number generator (PRNG) using a locally generated seed that is unlikely to be used by any other device in the same network. The PRNG **SHOULD** be cryptographically strong [RFC4086]. When allocating a new Session ID, GRASP **MUST** check that the value is not already in use and **SHOULD** check that it has not been used recently by consulting a cache of current and recent sessions. In the unlikely event of a clash, GRASP **MUST** generate a new value.

However, there is a finite probability that two nodes might generate the same Session ID value. For that reason, when a Session ID is communicated via GRASP, the receiving node **MUST** tag it with the initiator's IP address to allow disambiguation. In the highly unlikely event of two peers opening sessions with the same Session ID value, this tag will allow the two sessions to be distinguished. Multicast GRASP messages and their responses, which may be relayed between links, therefore include a field that carries the initiator's global IP address.

There is a highly unlikely race condition in which two peers start simultaneous negotiation sessions with each other using the same Session ID value. Depending on various implementation choices, this might lead to the two sessions being confused. See Section 2.8.6 for details of how to avoid this.

2.8. GRASP Messages

2.8.1. Message Overview

This section defines the GRASP message format and message types. Message types not listed here are reserved for future use.

The messages currently defined are:

Discovery and Discovery Response (M_DISCOVERY, M_RESPONSE).

Request Negotiation, Negotiation, Confirm Waiting, and Negotiation End (M_REQ_NEG, M_NEGOTIATE, M_WAIT, M_END).

Request Synchronization, Synchronization, and Flood Synchronization (M_REQ_SYN, M_SYNCH, M_FLOOD).

No Operation and Invalid (M_NOOP, M_INVALID).

2.8.2. GRASP Message Format

GRASP messages share an identical header format and a variable format area for options. GRASP message headers and options are transmitted in Concise Binary Object Representation (CBOR) [RFC8949]. In this specification, they are described using Concise Data Definition Language (CDDL) [RFC8610]. Fragmentary CDDL is used to describe each item in this section. A complete and normative CDDL specification of GRASP is given in Section 4, including constants such as message types.

Every GRASP message, except the No Operation message, carries a Session ID (Section 2.7). Options are then presented serially.

In fragmentary CDDL, every GRASP message follows the pattern:

```
grasp-message = (message .within message-structure) / noop-message

message-structure = [MESSAGE_TYPE, session-id, ?initiator,
                     *grasp-option]

MESSAGE_TYPE = 0..255
session-id = 0..4294967295 ; up to 32 bits
grasp-option = any
```

The MESSAGE_TYPE indicates the type of the message and thus defines the expected options. Any options received that are not consistent with the MESSAGE_TYPE **SHOULD** be silently discarded.

The No Operation (noop) message is described in Section 2.8.13.

The various MESSAGE_TYPE values are defined in Section 4.

All other message elements are described below and formally defined in Section 4.

If an unrecognized MESSAGE_TYPE is received in a unicast message, an Invalid message (Section 2.8.12) **MAY** be returned. Otherwise, the message **MAY** be logged and **MUST** be discarded. If an unrecognized MESSAGE_TYPE is received in a multicast message, it **MAY** be logged and **MUST** be silently discarded.

2.8.3. Message Size

GRASP nodes **MUST** be able to receive unicast messages of at least GRASP_DEF_MAX_SIZE bytes. GRASP nodes **MUST NOT** send unicast messages longer than GRASP_DEF_MAX_SIZE bytes unless a longer size is explicitly allowed for the objective concerned. For example, GRASP negotiation itself could be used to agree on a longer message size.

The message parser used by GRASP should be configured to know about the GRASP_DEF_MAX_SIZE, or any larger negotiated message size, so that it may defend against overly long messages.

The maximum size of multicast messages (M_DISCOVERY and M_FLOOD) depends on the link-layer technology or the link-adaptation layer in use.

2.8.4. Discovery Message

In fragmentary CDDL, a Discovery message follows the pattern:

```
discovery-message = [M_DISCOVERY, session-id, initiator, objective]
```

A discovery initiator sends a Discovery message to initiate a discovery process for a particular objective option.

The discovery initiator sends all Discovery messages via UDP to port GRASP_LISTEN_PORT at the link-local ALL_GRASP_NEIGHBORS multicast address on each link-layer interface in use by GRASP. It then listens for unicast TCP responses on a given port and stores the discovery results, including responding discovery objectives and corresponding unicast locators.

The listening port used for TCP **MUST** be the same port as used for sending the Discovery UDP multicast, on a given interface. In an implementation with a single GRASP instance in a node, this **MAY** be GRASP_LISTEN_PORT. To support multiple instances in the same node, the GRASP discovery mechanism in each instance needs to find, for each interface, a dynamic port that it can bind to for both sending UDP link-local multicast and listening for TCP before initiating any discovery.

The 'initiator' field in the message is a globally unique IP address of the initiator for the sole purpose of disambiguating the Session ID in other nodes. If for some reason the initiator does not have a globally unique IP address, it **MUST** use a link-local address that is highly likely to be unique for this purpose, for example, using [RFC7217]. Determination of a node's globally unique IP address is implementation dependent.

A Discovery message **MUST** include exactly one of the following:

- A Discovery Objective option (Section 2.10.1). Its loop count **MUST** be set to a suitable value to prevent discovery loops (default value is GRASP_DEF_LOOPCT). If the discovery initiator requires only on-link responses, the loop count **MUST** be set to 1.
- A Negotiation Objective option (Section 2.10.1). This is used both for the purpose of discovery and to indicate to the discovery target that it **MAY** directly reply to the discovery initiator with a Negotiation message for rapid processing, if it could act as the corresponding negotiation counterpart. The sender of such a Discovery message **MUST** initialize a negotiation timer and loop count in the same way as a Request Negotiation message (Section 2.8.6).
- A Synchronization Objective option (Section 2.10.1). This is used both for the purpose of discovery and to indicate to the discovery target that it **MAY** directly reply to the discovery initiator with a Synchronization message for rapid processing, if it could act as the corresponding synchronization counterpart. Its loop count **MUST** be set to a suitable value to prevent discovery loops (default value is GRASP_DEF_LOOPCT).

As mentioned in Section 2.5.4.2, a Discovery message **MAY** be sent unicast to a peer node, which **SHOULD** then proceed exactly as if the message had been multicast.

2.8.5. Discovery Response Message

In fragmentary CDDL, a Discovery Response message follows the pattern:

```
response-message = [M_RESPONSE, session-id, initiator, ttl,
                    (+locator-option // divert-option), ?objective]

ttl = 0..4294967295 ; in milliseconds
```

A node that receives a Discovery message **SHOULD** send a Discovery Response message if and only if it can respond to the discovery.

It **MUST** contain the same Session ID and initiator as the Discovery message.

It **MUST** contain a time-to-live (ttl) for the validity of the response, given as a positive integer value in milliseconds. Zero implies a value significantly greater than GRASP_DEF_TIMEOUT milliseconds (Section 2.6). A suggested value is ten times that amount.

It **MAY** include a copy of the discovery objective from the Discovery message.

It is sent to the sender of the Discovery message via TCP at the port used to send the Discovery message (as explained in Section 2.8.4). In the case of a relayed Discovery message, the Discovery Response is thus sent to the relay, not the original initiator.

In all cases, the transport session **SHOULD** be closed after sending the Discovery Response. A transport session failure is treated as no response.

If the responding node supports the discovery objective of the discovery, it **MUST** include at least one kind of locator option (Section 2.9.5) to indicate its own location. A sequence of multiple kinds of locator options (e.g., IP address option and FQDN option) is also valid.

If the responding node itself does not support the discovery objective, but it knows the locator of the discovery objective, then it **SHOULD** respond to the Discovery message with a Divert option (Section 2.9.2) embedding a locator option or a combination of multiple kinds of locator options that indicate the locator(s) of the discovery objective.

More details on the processing of Discovery Responses are given in Section 2.5.4.

2.8.6. Request Messages

In fragmentary CDDL, Request Negotiation and Request Synchronization messages follow the patterns:

```
request-negotiation-message = [M_REQ_NEG, session-id, objective]

request-synchronization-message = [M_REQ_SYN, session-id, objective]
```

A negotiation or synchronization requesting node sends the appropriate Request message to the unicast address of the negotiation or synchronization counterpart, using the appropriate protocol and port numbers (selected from the discovery result). If the discovery result is an FQDN, it will be resolved first.

A Request message **MUST** include the relevant objective option. In the case of Request Negotiation, the objective option **MUST** include the requested value.

When an initiator sends a Request Negotiation message, it **MUST** initialize a negotiation timer for the new negotiation thread. The default is GRASP_DEF_TIMEOUT milliseconds. Unless this timeout is modified by a Confirm Waiting message (Section 2.8.9), the initiator will consider that the negotiation has failed when the timer expires.

Similarly, when an initiator sends a Request Synchronization, it **SHOULD** initialize a synchronization timer. The default is GRASP_DEF_TIMEOUT milliseconds. The initiator will consider that synchronization has failed if there is no response before the timer expires.

When an initiator sends a Request message, it **MUST** initialize the loop count of the objective option with a value defined in the specification of the option or, if no such value is specified, with GRASP_DEF_LOOPCT.

If a node receives a Request message for an objective for which no ASA is currently listening, it **MUST** immediately close the relevant socket to indicate this to the initiator. This is to avoid unnecessary timeouts if, for example, an ASA exits prematurely but the GRASP core is listening on its behalf.

To avoid the highly unlikely race condition in which two nodes simultaneously request sessions with each other using the same Session ID (Section 2.7), a node **MUST** verify that the received Session ID is not already locally active when it receives a Request message. In case of a clash, it **MUST** discard the Request message, in which case the initiator will detect a timeout.

2.8.7. Negotiation Message

In fragmentary CDDL, a Negotiation message follows the pattern:

```
negotiation-message = [M_NEGOTIATE, session-id, objective]
```

A negotiation counterpart sends a Negotiation message in response to a Request Negotiation message, a Negotiation message, or a Discovery message in rapid mode. A negotiation process **MAY** include multiple steps.

The Negotiation message **MUST** include the relevant Negotiation Objective option, with its value updated according to progress in the negotiation. The sender **MUST** decrement the loop count by 1. If the loop count becomes zero, the message **MUST NOT** be sent. In this case, the negotiation session has failed and will time out.

2.8.8. Negotiation End Message

In fragmentary CDDL, a Negotiation End message follows the pattern:

```
end-message = [M_END, session-id, accept-option / decline-option]
```

A negotiation counterpart sends a Negotiation End message to close the negotiation. It **MUST** contain either an Accept option or a Decline option, defined in Section 2.9.3 and Section 2.9.4. It could be sent either by the requesting node or the responding node.

2.8.9. Confirm Waiting Message

In fragmentary CDDL, a Confirm Waiting message follows the pattern:

```
wait-message = [M_WAIT, session-id, waiting-time]
waiting-time = 0..4294967295 ; in milliseconds
```

A responding node sends a Confirm Waiting message to ask the requesting node to wait for a further negotiation response. It might be that the local process needs more time or that the negotiation depends on another triggered negotiation. This message **MUST NOT** include any other options. When received, the waiting time value overwrites and restarts the current negotiation timer (Section 2.8.6).

The responding node **SHOULD** send a Negotiation, Negotiation End, or another Confirm Waiting message before the negotiation timer expires. If not, when the initiator's timer expires, the initiator **MUST** treat the negotiation procedure as failed.

2.8.10. Synchronization Message

In fragmentary CDDL, a Synchronization message follows the pattern:

```
synch-message = [M_SYNCH, session-id, objective]
```

A node that receives a Request Synchronization, or a Discovery message in rapid mode, sends back a unicast Synchronization message with the synchronization data, in the form of a GRASP option for the specific synchronization objective present in the Request Synchronization.

2.8.11. Flood Synchronization Message

In fragmentary CDDL, a Flood Synchronization message follows the pattern:

```
flood-message = [M_FLOOD, session-id, initiator, ttl,
                 +[objective, (locator-option / [])]]

ttl = 0..4294967295 ; in milliseconds
```

A node **MAY** initiate flooding by sending an unsolicited Flood Synchronization message with synchronization data. This **MAY** be sent to port GRASP_LISTEN_PORT at the link-local ALL_GRASP_NEIGHBORS multicast address, in accordance with the rules in Section 2.5.6.

The initiator address is provided, as described for Discovery messages (Section 2.8.4), only to disambiguate the Session ID.

The message **MUST** contain a time-to-live (ttl) for the validity of the contents, given as a positive integer value in milliseconds. There is no default; zero indicates an indefinite lifetime.

The synchronization data are in the form of GRASP option(s) for specific synchronization objective(s). The loop count(s) **MUST** be set to a suitable value to prevent flood loops (default value is GRASP_DEF_LOOPCT).

Each objective option **MAY** be followed by a locator option (Section 2.9.5) associated with the flooded objective. In its absence, an empty option **MUST** be included to indicate a null locator.

A node that receives a Flood Synchronization message **MUST** cache the received objectives for use by local ASAs. Each cached objective **MUST** be tagged with the locator option sent with it, or with a null tag if an empty locator option was sent. If a subsequent Flood Synchronization message carries an objective with the same name and the same tag, the corresponding cached copy of the objective **MUST** be overwritten. If a subsequent Flood Synchronization message carrying an objective with same name arrives with a different tag, a new cached entry **MUST** be created.

Note: the purpose of this mechanism is to allow the recipient of flooded values to distinguish between different senders of the same objective, and if necessary communicate with them using the locator, protocol, and port included in the locator option. Many objectives will not need this mechanism, so they will be flooded with a null locator.

Cached entries **MUST** be ignored or deleted after their lifetime expires.

2.8.12. Invalid Message

In fragmentary CDDL, an Invalid message follows the pattern:

```
invalid-message = [M_INVALID, session-id, ?any]
```

This message **MAY** be sent by an implementation in response to an incoming unicast message that it considers invalid. The Session ID value **MUST** be copied from the incoming message. The content **SHOULD** be diagnostic information such as a partial copy of the invalid message up to the maximum message size. An M_INVALID message **MAY** be silently ignored by a recipient. However, it could be used in support of extensibility, since it indicates that the remote node does not support a new or obsolete message or option.

An M_INVALID message **MUST NOT** be sent in response to an M_INVALID message.

2.8.13. No Operation Message

In fragmentary CDDL, a No Operation message follows the pattern:

```
noop-message = [M_NOOP]
```

This message **MAY** be sent by an implementation that for practical reasons needs to initialize a socket. It **MUST** be silently ignored by a recipient.

2.9. GRASP Options

This section defines the GRASP options for the negotiation and synchronization protocol signaling. Additional options may be defined in the future.

2.9.1. Format of GRASP Options

GRASP options **SHOULD** be CBOR arrays that **MUST** start with an unsigned integer identifying the specific option type carried in this option. These option types are formally defined in Section 4.

GRASP options may be defined to include encapsulated GRASP options.

2.9.2. Divert Option

The Divert option is used to redirect a GRASP request to another node, which may be more appropriate for the intended negotiation or synchronization. It may redirect to an entity that is known as a specific negotiation or synchronization counterpart (on-link or off-link) or a default gateway. The Divert option **MUST** only be encapsulated in Discovery Response messages. If found elsewhere, it **SHOULD** be silently ignored.

A discovery initiator **MAY** ignore a Divert option if it only requires direct Discovery Responses.

In fragmentary CDDL, the Divert option follows the pattern:

```
divert-option = [O_DIVERT, +locator-option]
```

The embedded locator option(s) (Section 2.9.5) point to diverted destination target(s) in response to a Discovery message.

2.9.3. Accept Option

The Accept option is used to indicate to the negotiation counterpart that the proposed negotiation content is accepted.

The Accept option **MUST** only be encapsulated in Negotiation End messages. If found elsewhere, it **SHOULD** be silently ignored.

In fragmentary CDDL, the Accept option follows the pattern:

```
accept-option = [O_ACCEPT]
```

2.9.4. Decline Option

The Decline option is used to indicate to the negotiation counterpart the proposed negotiation content is declined and to end the negotiation process.

The Decline option **MUST** only be encapsulated in Negotiation End messages. If found elsewhere, it **SHOULD** be silently ignored.

In fragmentary CDDL, the Decline option follows the pattern:

```
decline-option = [O_DECLINE, ?reason]
reason = text  ; optional UTF-8 error message
```

Note: there might be scenarios where an ASA wants to decline the proposed value and restart the negotiation process. In this case, it is an implementation choice whether to send a Decline option or to continue with a Negotiation message, with an objective option that contains a null value or one that contains a new value that might achieve convergence.

2.9.5. Locator Options

These locator options are used to present reachability information for an ASA, a device, or an interface. They are Locator IPv6 Address option, Locator IPv4 Address option, Locator FQDN option, and Locator URI option.

Since ASAs will normally run as independent user programs, locator options need to indicate the network-layer locator plus the transport protocol and port number for reaching the target. For this reason, the locator options for IP addresses and FQDNs include this information explicitly. In the case of the Locator URI option, this information can be encoded in the URI itself.

Note: It is assumed that all locators used in locator options are in scope throughout the GRASP domain. As stated in Section 2.2, GRASP is not intended to work across disjoint addressing or naming realms.

2.9.5.1. Locator IPv6 Address Option

In fragmentary CDDL, the Locator IPv6 Address option follows the pattern:

```
ipv6-locator-option = [O_IPv6_LOCATOR, ipv6-address,
                       transport-proto, port-number]
ipv6-address = bytes .size 16

transport-proto = IPPROTO_TCP / IPPROTO_UDP
IPPROTO_TCP = 6
IPPROTO_UDP = 17
port-number = 0..65535
```

The content of this option is a binary IPv6 address followed by the protocol number and port number to be used.

Note 1: The IPv6 address **MUST** normally have global scope. However, during initialization, a link-local address **MAY** be used for specific objectives only (Section 2.5.2). In this case, the corresponding Discovery Response message **MUST** be sent via the interface to which the link-local address applies.

Note 2: A link-local IPv6 address **MUST NOT** be used when this option is included in a Divert option.

Note 3: The IPPROTO values are taken from the existing IANA Protocol Numbers registry in order to specify TCP or UDP. If GRASP requires future values that are not in that registry, a new registry for values outside the range 0..255 will be needed.

2.9.5.2. Locator IPv4 Address Option

In fragmentary CDDL, the Locator IPv4 Address option follows the pattern:

```
ipv4-locator-option = [O_IPv4_LOCATOR, ipv4-address,
                       transport-proto, port-number]
ipv4-address = bytes .size 4
```

The content of this option is a binary IPv4 address followed by the protocol number and port number to be used.

Note: If an operator has internal network address translation for IPv4, this option **MUST NOT** be used within the Divert option.

2.9.5.3. Locator FQDN Option

In fragmentary CDDL, the Locator FQDN option follows the pattern:

```
fqdn-locator-option = [O_FQDN_LOCATOR, text,
                       transport-proto, port-number]
```

The content of this option is the FQDN of the target followed by the protocol number and port number to be used.

Note 1: Any FQDN that might not be valid throughout the network in question, such as a Multicast DNS name [RFC6762], **MUST NOT** be used when this option is used within the Divert option.

Note 2: Normal GRASP operations are not expected to use this option. It is intended for special purposes such as discovering external services.

2.9.5.4. Locator URI Option

In fragmentary CDDL, the Locator URI option follows the pattern:

```
uri-locator-option = [O_URI_LOCATOR, text,
                      transport-proto / null, port-number / null]
```

The content of this option is the URI of the target followed by the protocol number and port number to be used (or by null values if not required) [RFC3986].

Note 1: Any URI which might not be valid throughout the network in question, such as one based on a Multicast DNS name [RFC6762], **MUST NOT** be used when this option is used within the Divert option.

Note 2: Normal GRASP operations are not expected to use this option. It is intended for special purposes such as discovering external services. Therefore, its use is not further described in this specification.

2.10. Objective Options

2.10.1. Format of Objective Options

An objective option is used to identify objectives for the purposes of discovery, negotiation, or synchronization. All objectives **MUST** be in the following format, described in fragmentary CDDL:

```
objective = [objective-name, objective-flags,
             loop-count, ?objective-value]

objective-name = text
objective-value = any
loop-count = 0..255
```

All objectives are identified by a unique name that is a UTF-8 string [RFC3629], to be compared byte by byte.

The names of generic objectives **MUST NOT** include a colon (":") and **MUST** be registered with IANA (Section 5).

The names of privately defined objectives **MUST** include at least one colon (":"). The string preceding the last colon in the name **MUST** be globally unique and in some way identify the entity or person defining the objective. The following three methods **MAY** be used to create such a globally unique string:

1. The unique string is a decimal number representing a registered 32-bit Private Enterprise Number (PEN) [RFC5612] that uniquely identifies the enterprise defining the objective.
2. The unique string is a FQDN that uniquely identifies the entity or person defining the objective.
3. The unique string is an email address that uniquely identifies the entity or person defining the objective.

GRASP treats the objective name as an opaque string. For example, "EX1", "32473:EX1", "example.com:EX1", "example.org:EX1", and "user@example.org:EX1" are five different objectives.

The 'objective-flags' field is described in Section 2.10.2.

The 'loop-count' field is used for terminating negotiation as described in Section 2.8.7. It is also used for terminating discovery as described in Section 2.5.4 and for terminating flooding as described in Section 2.5.6.2. It is placed in the objective rather than in the GRASP message format because, as far as the ASA is concerned, it is a property of the objective itself.

The 'objective-value' field expresses the actual value of a negotiation or synchronization objective. Its format is defined in the specification of the objective and may be a simple value or a data structure of any kind, as long as it can be represented in CBOR. It is optional only in a Discovery or Discovery Response message.

2.10.2. Objective Flags

An objective may be relevant for discovery only, for discovery and negotiation, or for discovery and synchronization. This is expressed in the objective by logical flag bits:

```
objective-flags = uint .bits objective-flag
objective-flag = &(
  F_DISC: 0    ; valid for discovery
  F_NEG: 1     ; valid for negotiation
  F_SYNCH: 2   ; valid for synchronization
  F_NEG_DRY: 3 ; negotiation is a dry run
)
```

These bits are independent and may be combined appropriately, e.g., (F_DISC and F_SYNCH) or (F_DISC and F_NEG) or (F_DISC and F_NEG and F_NEG_DRY).

Note that for a given negotiation session, an objective must be used either for negotiation or for dry-run negotiation. Mixing the two modes in a single negotiation is not possible.

2.10.3. General Considerations for Objective Options

As mentioned above, objective options **MUST** be assigned a unique name. As long as privately defined objective options obey the rules above, this document does not restrict their choice of name, but the entity or person concerned **SHOULD** publish the names in use.

Names are expressed as UTF-8 strings for convenience in designing objective options for localized use. For generic usage, names expressed in the ASCII subset of UTF-8 are **RECOMMENDED**. Designers planning to use non-ASCII names are strongly advised to consult [RFC8264] or its successor to understand the complexities involved. Since GRASP compares names byte by byte, all issues of Unicode profiling and canonicalization **MUST** be specified in the design of the objective option.

All objective options **MUST** respect the CBOR patterns defined above as "objective" and **MUST** replace the 'any' field with a valid CBOR data definition for the relevant use case and application.

An objective option that contains no additional fields beyond its 'loop-count' can only be a discovery objective and **MUST** only be used in Discovery and Discovery Response messages.

The Negotiation Objective options contain negotiation objectives, which vary according to different functions and/or services. They **MUST** be carried by Discovery, Request Negotiation, or Negotiation messages only. The negotiation initiator **MUST** set the initial 'loop-count' to a value specified in the specification of the objective or, if no such value is specified, to GRASP_DEF_LOOPCT.

For most scenarios, there should be initial values in the negotiation requests. Consequently, the Negotiation Objective options **MUST** always be completely presented in a Request Negotiation message, or in a Discovery message in rapid mode. If there is no initial value, the 'value' field **SHOULD** be set to the 'null' value defined by CBOR.

Synchronization Objective options are similar, but **MUST** be carried by Discovery, Discovery Response, Request Synchronization, or Flood Synchronization messages only. They include 'value' fields only in Synchronization or Flood Synchronization messages.

The design of an objective interacts in various ways with the design of the ASAs that will use it. ASA design considerations are discussed in [ASA-GUIDELINES].

2.10.4. Organizing of Objective Options

Generic objective options **MUST** be specified in documents available to the public and **SHOULD** be designed to use either the negotiation or the synchronization mechanism described above.

As noted earlier, one negotiation objective is handled by each GRASP negotiation thread. Therefore, a negotiation objective, which is based on a specific function or action, **SHOULD** be organized as a single GRASP option. It is **NOT RECOMMENDED** to organize multiple negotiation objectives into a single option nor to split a single function or action into multiple negotiation objectives.

It is important to understand that GRASP negotiation does not support transactional integrity. If transactional integrity is needed for a specific objective, this must be ensured by the ASA. For example, an ASA might need to ensure that it only participates in one negotiation thread at the same time. Such an ASA would need to stop listening for incoming negotiation requests before generating an outgoing negotiation request.

A synchronization objective **SHOULD** be organized as a single GRASP option.

Some objectives will support more than one operational mode. An example is a negotiation objective with both a dry-run mode (where the negotiation is to determine whether the other end can, in fact, make the requested change without problems) and a live mode, as explained in Section 2.5.5. The semantics of such modes will be defined in the specification of the objectives. These objectives **SHOULD** include flags indicating the applicable mode(s).

An issue requiring particular attention is that GRASP itself is not a transactionally safe protocol. Any state associated with a dry-run operation, such as temporarily reserving a resource for subsequent use in a live run, is entirely a matter for the designer of the ASA concerned.

As indicated in Section 2.1, an objective's value may include multiple parameters. Parameters might be categorized into two classes: the obligatory ones presented as fixed fields and the optional ones presented in some other form of data structure embedded in CBOR. The format might be inherited from an existing management or configuration protocol, with the objective option acting as a carrier for that format. The data structure might be defined in a formal language, but that is a matter for the specifications of individual objectives. There are many candidates, according to the context, such as ABNF, RBNF, XML Schema, YANG, etc. GRASP itself is agnostic on these questions. The only restriction is that the format can be mapped into CBOR.

It is **NOT RECOMMENDED** to mix parameters that have significantly different response-time characteristics in a single objective. Separate objectives are more suitable for such a scenario.

All objectives **MUST** support GRASP discovery. However, as mentioned in Section 2.3, it is acceptable for an ASA to use an alternative method of discovery.

Normally, a GRASP objective will refer to specific technical parameters as explained in Section 2.1. However, it is acceptable to define an abstract objective for the purpose of managing or coordinating ASAs. It is also acceptable to define a special-purpose objective for purposes such as trust bootstrapping or formation of the ACP.

To guarantee convergence, a limited number of rounds or a timeout is needed for each negotiation objective. Therefore, the definition of each negotiation objective **SHOULD** clearly specify this, for example, a default loop count and timeout, so that the negotiation can always be terminated properly. If not, the GRASP defaults will apply.

There must be a well-defined procedure for concluding that a negotiation cannot succeed, and if so, deciding what happens next (e.g., deadlock resolution, tie-breaking, or reversion to best-effort service). This **MUST** be specified for individual negotiation objectives.

2.10.5. Experimental and Example Objective Options

The names "EX0" through "EX9" have been reserved for experimental options. Multiple names have been assigned because a single experiment may use multiple options simultaneously. These experimental options are highly likely to have different meanings when used for different experiments. Therefore, they **SHOULD NOT** be used without an explicit human decision and **MUST NOT** be used in unmanaged networks such as home networks.

These names are also **RECOMMENDED** for use in documentation examples.

3. Security Considerations

A successful attack on negotiation-enabled nodes would be extremely harmful, as such nodes might end up with a completely undesirable configuration that would also adversely affect their peers. GRASP nodes and messages therefore require full protection. As explained in Section 2.5.1, GRASP **MUST** run within a secure environment such as the ACP [RFC8994], except for the constrained instances described in Section 2.5.2.

Authentication
: A cryptographically authenticated identity for each device is needed in an Autonomic Network. It is not safe to assume that a large network is physically secured against interference or that all personnel are trustworthy. Each autonomic node **MUST** be capable of proving its identity and authenticating its messages. GRASP relies on a separate, external certificate-based security mechanism to support authentication, data integrity protection, and anti-replay protection.

  Since GRASP must be deployed in an existing secure environment, the protocol itself specifies nothing concerning the trust anchor and certification authority. For example, in the ACP [RFC8994], all nodes can trust each other and the ASAs installed in them.

  If GRASP is used temporarily without an external security mechanism, for example, during system bootstrap (Section 2.5.1), the Session ID (Section 2.7) will act as a nonce to provide limited protection against the injecting of responses by third parties. A full analysis of the secure bootstrap process is in [RFC8995].

Authorization and roles
: GRASP is agnostic about the roles and capabilities of individual ASAs and about which objectives a particular ASA is authorized to support. An implementation might support precautions such as allowing only one ASA in a given node to modify a given objective, but this may not be appropriate in all cases. For example, it might be operationally useful to allow an old and a new version of the same ASA to run simultaneously during an overlap period. These questions are out of scope for the present specification.

Privacy and confidentiality
: GRASP is intended for network-management purposes involving network elements, not end hosts. Therefore, no personal information is expected to be involved in the signaling protocol, so there should be no direct impact on personal privacy. Nevertheless, applications that do convey personal information cannot be excluded. Also, traffic flow paths, VPNs, etc., could be negotiated, which could be of interest for traffic analysis. Operators generally want to conceal details of their network topology and traffic density from outsiders. Therefore, since insider attacks cannot be excluded in a large network, the security mechanism for the protocol **MUST** provide message confidentiality. This is why Section 2.5.1 requires either an ACP or an alternative security mechanism.

Link-local multicast security
: GRASP has no reasonable alternative to using link-local multicast for Discovery or Flood Synchronization messages, and these messages are sent in the clear and with no authentication. They are only sent on interfaces within the Autonomic Network (see Section 2.1 and Section 2.5.1). They are, however, available to on-link eavesdroppers and could be forged by on-link attackers. In the case of discovery, the Discovery Responses are unicast and will therefore be protected (Section 2.5.1), and an untrusted forger will not be able to receive responses. In the case of flood synchronization, an on-link eavesdropper will be able to receive the flooded objectives, but there is no response message to consider. Some precautions for Flood Synchronization messages are suggested in Section 2.5.6.2.

DoS attack protection
: GRASP discovery partly relies on insecure link-local multicast. Since routers participating in GRASP sometimes relay Discovery messages from one link to another, this could be a vector for denial-of-service attacks. Some mitigations are specified in Section 2.5.4. However, malicious code installed inside the ACP could always launch DoS attacks consisting of either spurious Discovery messages or spurious Discovery Responses. It is important that firewalls prevent any GRASP messages from entering the domain from an unknown source.

Security during bootstrap and discovery
: A node cannot trust GRASP traffic from other nodes until the security environment (such as the ACP) has identified the trust anchor and can authenticate traffic by validating certificates for other nodes. Also, until it has successfully enrolled [RFC8995], a node cannot assume that other nodes are able to authenticate its own traffic. Therefore, GRASP discovery during the bootstrap phase for a new device will inevitably be insecure. Secure synchronization and negotiation will be impossible until enrollment is complete. Further details are given in Section 2.5.2.

Security of discovered locators
: When GRASP discovery returns an IP address, it **MUST** be that of a node within the secure environment (Section 2.5.1). If it returns an FQDN or a URI, the ASA that receives it **MUST NOT** assume that the target of the locator is within the secure environment.

4. CDDL Specification of GRASP

```
<CODE BEGINS> file "grasp.cddl"

grasp-message = (message .within message-structure) / noop-message

message-structure = [MESSAGE_TYPE, session-id, ?initiator,
                     *grasp-option]

MESSAGE_TYPE = 0..255
session-id = 0..4294967295 ; up to 32 bits
grasp-option = any

message /= discovery-message
discovery-message = [M_DISCOVERY, session-id, initiator, objective]

message /= response-message ; response to Discovery
response-message = [M_RESPONSE, session-id, initiator, ttl,
                    (+locator-option // divert-option), ?objective]

message /= synch-message ; response to Synchronization request
synch-message = [M_SYNCH, session-id, objective]

message /= flood-message
flood-message = [M_FLOOD, session-id, initiator, ttl,
                 +[objective, (locator-option / [])]]

message /= request-negotiation-message
request-negotiation-message = [M_REQ_NEG, session-id, objective]

message /= request-synchronization-message
request-synchronization-message = [M_REQ_SYN, session-id, objective]

message /= negotiation-message
negotiation-message = [M_NEGOTIATE, session-id, objective]

message /= end-message
end-message = [M_END, session-id, accept-option / decline-option]

message /= wait-message
wait-message = [M_WAIT, session-id, waiting-time]

message /= invalid-message
invalid-message = [M_INVALID, session-id, ?any]

noop-message = [M_NOOP]

divert-option = [O_DIVERT, +locator-option]

accept-option = [O_ACCEPT]

decline-option = [O_DECLINE, ?reason]
reason = text  ; optional UTF-8 error message

waiting-time = 0..4294967295 ; in milliseconds
ttl = 0..4294967295 ; in milliseconds

locator-option /= [O_IPv4_LOCATOR, ipv4-address,
                   transport-proto, port-number]
```

```
ipv4-address = bytes .size 4

locator-option /= [O_IPv6_LOCATOR, ipv6-address,
                   transport-proto, port-number]
ipv6-address = bytes .size 16

locator-option /= [O_FQDN_LOCATOR, text, transport-proto,
                   port-number]

locator-option /= [O_URI_LOCATOR, text,
                   transport-proto / null, port-number / null]

transport-proto = IPPROTO_TCP / IPPROTO_UDP
IPPROTO_TCP = 6
IPPROTO_UDP = 17
port-number = 0..65535

initiator = ipv4-address / ipv6-address

objective-flags = uint .bits objective-flag

objective-flag = &(
  F_DISC: 0    ; valid for discovery
  F_NEG: 1     ; valid for negotiation
  F_SYNCH: 2   ; valid for synchronization
  F_NEG_DRY: 3 ; negotiation is a dry run
)

objective = [objective-name, objective-flags,
             loop-count, ?objective-value]

objective-name = text ; see section "Format of Objective Options"

objective-value = any

loop-count = 0..255

; Constants for message types and option types

M_NOOP = 0
M_DISCOVERY = 1
M_RESPONSE = 2
M_REQ_NEG = 3
M_REQ_SYN = 4
M_NEGOTIATE = 5
M_END = 6
M_WAIT = 7
M_SYNCH = 8
M_FLOOD = 9
M_INVALID = 99

O_DIVERT = 100
O_ACCEPT = 101
O_DECLINE = 102
O_IPv6_LOCATOR = 103
O_IPv4_LOCATOR = 104
O_FQDN_LOCATOR = 105
O_URI_LOCATOR = 106
```

```
<CODE ENDS>
```

5. IANA Considerations

This document defines the GeneRic Autonomic Signaling Protocol (GRASP).

Section 2.6 explains the following link-local multicast addresses that IANA has assigned for use by GRASP.

Assigned in the "Link-Local Scope Multicast Addresses" subregistry of the "IPv6 Multicast Address Space Registry":

Address(es): ff02::13
Description: ALL_GRASP_NEIGHBORS
Reference: RFC 8990

Assigned in the "Local Network Control Block (224.0.0.0 - 224.0.0.255 (224.0.0/24))" subregistry of the "IPv4 Multicast Address Space Registry":

Address(es): 224.0.0.119
Description: ALL_GRASP_NEIGHBORS
Reference: RFC 8990

Section 2.6 explains the following User Port (GRASP_LISTEN_PORT), which IANA has assigned for use by GRASP for both UDP and TCP:

Service Name: grasp
Port Number: 7017
Transport Protocol: udp, tcp
Description GeneRic Autonomic Signaling Protocol
Assignee: IESG <iesg@ietf.org>
Contact: IETF Chair <chair@ietf.org>
Reference: RFC 8990

The IANA has created the "GeneRic Autonomic Signaling Protocol (GRASP) Parameters" registry, which includes two subregistries: "GRASP Messages and Options" and "GRASP Objective Names".

The values in the "GRASP Messages and Options" subregistry are names paired with decimal integers. Future values **MUST** be assigned using the Standards Action policy defined by [RFC8126]. The following initial values are assigned by this document:

Value	Message/Option
0	M_NOOP

Value	Message/Option
1	M_DISCOVERY
2	M_RESPONSE
3	M_REQ_NEG
4	M_REQ_SYN
5	M_NEGOTIATE
6	M_END
7	M_WAIT
8	M_SYNCH
9	M_FLOOD
99	M_INVALID
100	O_DIVERT
101	O_ACCEPT
102	O_DECLINE
103	O_IPv6_LOCATOR
104	O_IPv4_LOCATOR
105	O_FQDN_LOCATOR
106	O_URI_LOCATOR

Table 1: Initial Values of the "GRASP Messages and Options" Subregistry

The values in the "GRASP Objective Names" subregistry are UTF-8 strings that **MUST NOT** include a colon (":"), according to Section 2.10.1. Future values **MUST** be assigned using the Specification Required policy defined by [RFC8126].

To assist expert review of a new objective, the specification should include a precise description of the format of the new objective, with sufficient explanation of its semantics to allow independent implementations. See Section 2.10.3 for more details. If the new objective is similar in name or purpose to a previously registered objective, the specification should explain why a new objective is justified.

The following initial values are assigned by this document:

Objective Name	Reference
EX0	RFC 8990
EX1	RFC 8990
EX2	RFC 8990
EX3	RFC 8990
EX4	RFC 8990
EX5	RFC 8990
EX6	RFC 8990
EX7	RFC 8990
EX8	RFC 8990
EX9	RFC 8990

Table 2: Initial Values of the "GRASP Objective Names" Subregistry

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC7217] Gont, F., "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)", RFC 7217, DOI 10.17487/RFC7217, April 2014, <https://www.rfc-editor.org/info/rfc7217>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC8994] Eckert, T., Ed., Behringer, M., Ed., and S. Bjarnason, "An Autonomic Control Plane (ACP)", RFC 8994, DOI 10.17487/RFC8994, May 2021, <https://www.rfc-editor.org/info/rfc8994>.

6.2. Informative References

[ADNCP] Stenberg, M., "Autonomic Distributed Node Consensus Protocol", Work in Progress, Internet-Draft, draft-stenberg-anima-adncp-00, 5 March 2015, <https://tools.ietf.org/html/draft-stenberg-anima-adncp-00>.

[ASA-GUIDELINES] Carpenter, B., Ciavaglia, L., Jiang, S., and P. Peloso, "Guidelines for Autonomic Service Agents", Work in Progress, Internet-Draft, draft-ietf-anima-asa-guidelines-00, 14 November 2020, <https://tools.ietf.org/html/draft-ietf-anima-asa-guidelines-00>.

[IGCP] Behringer, M. H., Chaparadza, R., Xin, L., Mahkonen, H., and R. Petre, "IP based Generic Control Protocol (IGCP)", Work in Progress, Internet-Draft, draft-chaparadza-intarea-igcp-00, 25 July 2011, <https://tools.ietf.org/html/draft-chaparadza-intarea-igcp-00>.

[RFC2205] Braden, R., Ed., Zhang, L., Berson, S., Herzog, S., and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, DOI 10.17487/RFC2205, September 1997, <https://www.rfc-editor.org/info/rfc2205>.

[RFC2334] Luciani, J., Armitage, G., Halpern, J., and N. Doraswamy, "Server Cache Synchronization Protocol (SCSP)", RFC 2334, DOI 10.17487/RFC2334, April 1998, <https://www.rfc-editor.org/info/rfc2334>.

[RFC2608] Guttman, E., Perkins, C., Veizades, J., and M. Day, "Service Location Protocol, Version 2", RFC 2608, DOI 10.17487/RFC2608, June 1999, <https://www.rfc-editor.org/info/rfc2608>.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <https://www.rfc-editor.org/info/rfc2865>.

[RFC3416] Presuhn, R., Ed., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, DOI 10.17487/RFC3416, December 2002, <https://www.rfc-editor.org/info/rfc3416>.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, DOI 10.17487/RFC3493, February 2003, <https://www.rfc-editor.org/info/rfc3493>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <https://www.rfc-editor.org/info/rfc4861>.

[RFC5612] Eronen, P. and D. Harrington, "Enterprise Number for Documentation Use", RFC 5612, DOI 10.17487/RFC5612, August 2009, <https://www.rfc-editor.org/info/rfc5612>.

[RFC5971] Schulzrinne, H. and R. Hancock, "GIST: General Internet Signalling Transport", RFC 5971, DOI 10.17487/RFC5971, October 2010, <https://www.rfc-editor.org/info/rfc5971>.

[RFC6206] Levis, P., Clausen, T., Hui, J., Gnawali, O., and J. Ko, "The Trickle Algorithm", RFC 6206, DOI 10.17487/RFC6206, March 2011, <https://www.rfc-editor.org/info/rfc6206>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6733] Fajardo, V., Ed., Arkko, J., Loughney, J., and G. Zorn, Ed., "Diameter Base Protocol", RFC 6733, DOI 10.17487/RFC6733, October 2012, <https://www.rfc-editor.org/info/rfc6733>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/info/rfc6762>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC6887] Wing, D., Ed., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, DOI 10.17487/RFC6887, April 2013, <https://www.rfc-editor.org/info/rfc6887>.

[RFC7558] Lynn, K., Cheshire, S., Blanchet, M., and D. Migault, "Requirements for Scalable DNS-Based Service Discovery (DNS-SD) / Multicast DNS (mDNS) Extensions", RFC 7558, DOI 10.17487/RFC7558, July 2015, <https://www.rfc-editor.org/info/rfc7558>.

[RFC7575] Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, DOI 10.17487/RFC7575, June 2015, <https://www.rfc-editor.org/info/rfc7575>.

[RFC7576] Jiang, S., Carpenter, B., and M. Behringer, "General Gap Analysis for Autonomic Networking", RFC 7576, DOI 10.17487/RFC7576, June 2015, <https://www.rfc-editor.org/info/rfc7576>.

[RFC7787] Stenberg, M. and S. Barth, "Distributed Node Consensus Protocol", RFC 7787, DOI 10.17487/RFC7787, April 2016, <https://www.rfc-editor.org/info/rfc7787>.

[RFC7788] Stenberg, M., Barth, S., and P. Pfister, "Home Networking Control Protocol", RFC 7788, DOI 10.17487/RFC7788, April 2016, <https://www.rfc-editor.org/info/rfc7788>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8264] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 8264, DOI 10.17487/RFC8264, October 2017, <https://www.rfc-editor.org/info/rfc8264>.

[RFC8368] Eckert, T., Ed. and M. Behringer, "Using an Autonomic Control Plane for Stable Connectivity of Network Operations, Administration, and Maintenance (OAM)", RFC 8368, DOI 10.17487/RFC8368, May 2018, <https://www.rfc-editor.org/info/rfc8368>.

[RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <https://www.rfc-editor.org/info/rfc8415>.

[RFC8991] Carpenter, B., Liu, B., Ed., Wang, W., and X. Gong, "GeneRic Autonomic Signaling Protocol Application Program Interface (GRASP API)", RFC 8991, DOI 10.17487/RFC8991, May 2021, <https://www.rfc-editor.org/info/rfc8991>.

[RFC8993] Behringer, M., Ed., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", RFC 8993, DOI 10.17487/RFC8993, May 2021, <https://www.rfc-editor.org/info/rfc8993>.

[RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <https://www.rfc-editor.org/info/rfc8995>.

Appendix A. Example Message Formats

For readers unfamiliar with CBOR, this appendix shows a number of example GRASP messages conforming to the CDDL syntax given in Section 4. Each message is shown three times in the following formats:

1. CBOR diagnostic notation.
2. Similar, but showing the names of the constants. (Details of the flag bit encoding are omitted.)
3. Hexadecimal version of the CBOR wire format.

Long lines are split for display purposes only.

A.1. Discovery Example

The initiator (2001:db8:f000:baaa:28cc:dc4c:9703:6781) multicasts a Discovery message looking for objective EX1:

```
[1, 13948744, h'20010db8f000baaa28ccdc4c97036781', ["EX1", 5, 2, 0]]
[M_DISCOVERY, 13948744, h'20010db8f000baaa28ccdc4c97036781',
              ["EX1", F_SYNCH_bits, 2, 0]]
h'84011a00d4d7485020010db8f000baaa28ccdc4c9703678184634558310502 00'
```

A peer (2001:0db8:f000:baaa:f000:baaa:f000:baaa) responds with a locator:

```
[2, 13948744, h'20010db8f000baaa28ccdc4c97036781', 60000,
              [103, h'20010db8f000baaaf000baaaf000baaa', 6, 49443]]
[M_RESPONSE, 13948744, h'20010db8f000baaa28ccdc4c97036781', 60000,
              [O_IPv6_LOCATOR, h'20010db8f000baaaf000baaaf000baaa',
               IPPROTO_TCP, 49443]]
h'85021a00d4d7485020010db8f000baaa28ccdc4c9703678119ea6084186750
  20010db8f000baaaf000baaaf000baaa0619c123'
```

A.2. Flood Example

The initiator multicasts a Flood Synchronization message. The single objective has a null locator. There is no response:

```
[9, 3504974, h'20010db8f000baaa28ccdc4c97036781', 10000,
             [["EX1", 5, 2, ["Example 1 value=", 100]],[] ] ]
[M_FLOOD, 3504974, h'20010db8f000baaa28ccdc4c97036781', 10000,
             [["EX1", F_SYNCH_bits, 2, ["Example 1 value=", 100]],[] ] ]
h'85091a00357b4e5020010db8f000baaa28ccdc4c97036781192710
  828463455831050282704578616d706c6520312076616c75653d186480'
```

A.3. Synchronization Example

Following successful discovery of objective EX2, the initiator unicasts a Request Synchronization message:

```
[4, 4038926, ["EX2", 5, 5, 0]]
[M_REQ_SYN, 4038926, ["EX2", F_SYNCH_bits, 5, 0]]
h'83041a003da10e8463455832050500'
```

The peer responds with a value:

```
[8, 4038926, ["EX2", 5, 5, ["Example 2 value=", 200]]]
[M_SYNCH, 4038926, ["EX2", F_SYNCH_bits, 5, ["Example 2 value=", 200]]]
h'83081a003da10e8463455832050582704578616d706c6520322076616c75653d18c8'
```

A.4. Simple Negotiation Example

Following successful discovery of objective EX3, the initiator unicasts a Request Negotiation message:

```
[3, 802813, ["EX3", 3, 6, ["NZD", 47]]]
[M_REQ_NEG, 802813, ["EX3", F_NEG_bits, 6, ["NZD", 47]]]
h'83031a000c3ffd8463455833030682634e5a44182f'
```

The peer responds with immediate acceptance. Note that no objective is needed because the initiator's request was accepted without change:

```
[6, 802813, [101]]
[M_END , 802813, [O_ACCEPT]]
h'83061a000c3ffd811865'
```

A.5. Complete Negotiation Example

Again the initiator unicasts a Request Negotiation message:

```
[3, 13767778, ["EX3", 3, 6, ["NZD", 410]]]
[M_REQ_NEG, 13767778, ["EX3", F_NEG_bits, 6, ["NZD", 410]]]
h'83031a00d21462846345583303068263 4e5a4419019a'
```

The responder starts to negotiate (making an offer):

```
[5, 13767778, ["EX3", 3, 6, ["NZD", 80]]]
[M_NEGOTIATE, 13767778, ["EX3", F_NEG_bits, 6, ["NZD", 80]]]
h'83051a00d21462846345583303068263 4e5a441850'
```

The initiator continues to negotiate (reducing its request, and note that the loop count is decremented):

```
[5, 13767778, ["EX3", 3, 5, ["NZD", 307]]]
[M_NEGOTIATE, 13767778, ["EX3", F_NEG_bits, 5, ["NZD", 307]]]
h'83051a00d21462846345583030582634e5a44190133'
```

The responder asks for more time:

```
[7, 13767778, 34965]
[M_WAIT, 13767778, 34965]
h'83071a00d21462198895'
```

The responder continues to negotiate (increasing its offer):

```
[5, 13767778, ["EX3", 3, 4, ["NZD", 120]]]
[M_NEGOTIATE, 13767778, ["EX3", F_NEG_bits, 4, ["NZD", 120]]]
h'83051a00d21462846345583030482634e5a441878'
```

The initiator continues to negotiate (reducing its request):

```
[5, 13767778, ["EX3", 3, 3, ["NZD", 246]]]
[M_NEGOTIATE, 13767778, ["EX3", F_NEG_bits, 3, ["NZD", 246]]]
h'83051a00d21462846345583030382634e5a4418f6'
```

The responder refuses to negotiate further:

```
[6, 13767778, [102, "Insufficient funds"]]
[M_END , 13767778, [O_DECLINE, "Insufficient funds"]]
h'83061a00d2146282186672496e73756666696369656e742066756e6473'
```

This negotiation has failed. If either side had sent [M_END, 13767778, [O_ACCEPT]] it would have succeeded, converging on the objective value in the preceding M_NEGOTIATE. Note that apart from the initial M_REQ_NEG, the process is symmetrical.

Appendix B. Requirement Analysis of Discovery, Synchronization, and Negotiation

This section discusses the requirements for discovery, negotiation, and synchronization capabilities. The primary user of the protocol is an Autonomic Service Agent (ASA), so the requirements are mainly expressed as the features needed by an ASA. A single physical device might contain several ASAs, and a single ASA might manage several technical objectives. If a technical objective is managed by several ASAs, any necessary coordination is outside the scope of GRASP. Furthermore, requirements for ASAs themselves, such as the processing of Intent [RFC7575], are out of scope for the present document.

B.1. Requirements for Discovery

D1. ASAs may be designed to manage any type of configurable device or software, as required in Appendix B.2. A basic requirement is therefore that the protocol can represent and discover any kind of technical objective (as defined in Section 2.1) among arbitrary subsets of participating nodes.

In an Autonomic Network, we must assume that when a device starts up, it has no information about any peer devices, the network structure, or the specific role it must play. The ASA(s) inside the device are in the same situation. In some cases, when a new application session starts within a device, the device or ASA may again lack information about relevant peers. For example, it might be necessary to set up resources on multiple other devices, coordinated and matched to each other so that there is no wasted resource. Security settings might also need updating to allow for the new device or user. The relevant peers may be different for different technical objectives. Therefore discovery needs to be repeated as often as necessary to find peers capable of acting as counterparts for each objective that a discovery initiator needs to handle. From this background we derive the next three requirements:

D2. When an ASA first starts up, it may have no knowledge of the specific network to which it is attached. Therefore the discovery process must be able to support any network scenario, assuming only that the device concerned is bootstrapped from factory condition.

D3. When an ASA starts up, it must require no configured location information about any peers in order to discover them.

D4. If an ASA supports multiple technical objectives, relevant peers may be different for different discovery objectives, so discovery needs to be performed separately to find counterparts for each objective. Thus, there must be a mechanism by which an ASA can separately discover peer ASAs for each of the technical objectives that it needs to manage, whenever necessary.

D5. Following discovery, an ASA will normally perform negotiation or synchronization for the corresponding objectives. The design should allow for this by conveniently linking discovery to negotiation and synchronization. It may provide an optional mechanism to combine discovery and negotiation/synchronization in a single protocol exchange.

D6. Some objectives may only be significant on the local link, but others may be significant across the routed network and require off-link operations. Thus, the relevant peers might be immediate neighbors on the same layer 2 link, or they might be more distant and only accessible via layer 3. The mechanism must therefore provide both on-link and off-link discovery of ASAs supporting specific technical objectives.

D7. The discovery process should be flexible enough to allow for special cases, such as the following:

- During initialization, a device must be able to establish mutual trust with autonomic nodes elsewhere in the network and participate in an authentication mechanism. Although this will inevitably start with a discovery action, it is a special case precisely because trust is not yet established. This topic is the subject of [RFC8995]. We require that once trust has been established for a device, all ASAs within the device inherit the device's credentials and are also trusted. This does not preclude the device having multiple credentials.
- Depending on the type of network involved, discovery of other central functions might be needed, such as the Network Operations Center (NOC) [RFC8368]. The protocol must be capable of supporting such discovery during initialization, as well as discovery during ongoing operation.

D8. The discovery process must not generate excessive traffic and must take account of sleeping nodes.

D9. There must be a mechanism for handling stale discovery results.

B.2. Requirements for Synchronization and Negotiation Capability

Autonomic Networks need to be able to manage many different types of parameters and consider many dimensions, such as latency, load, unused or limited resources, conflicting resource requests, security settings, power saving, load balancing, etc. Status information and resource metrics need to be shared between nodes for dynamic adjustment of resources and for monitoring purposes. While this might be achieved by existing protocols when they are available, the new protocol needs to be able to support parameter exchange, including mutual synchronization, even when no negotiation as such is required. In general, these parameters do not apply to all participating nodes, but only to a subset.

SN1. A basic requirement for the protocol is therefore the ability to represent, discover, synchronize, and negotiate almost any kind of network parameter among selected subsets of participating nodes.

SN2. Negotiation is an iterative request/response process that must be guaranteed to terminate (with success or failure). While tie-breaking rules must be defined specifically for each use case, the protocol should have some general mechanisms in support of loop and deadlock prevention, such as hop-count limits or timeouts.

SN3. Synchronization must be possible for groups of nodes ranging from small to very large.

SN4. To avoid "reinventing the wheel", the protocol should be able to encapsulate the data formats used by existing configuration protocols (such as Network Configuration Protocol (NETCONF) and YANG) in cases where that is convenient.

SN5. Human intervention in complex situations is costly and error prone. Therefore, synchronization or negotiation of parameters without human intervention is desirable whenever the coordination of multiple devices can improve overall network performance. It follows that the protocol's resource requirements must be small enough to fit in any device that would otherwise need human intervention. The issue of running in constrained nodes is discussed in [RFC8993].

SN6. Human intervention in large networks is often replaced by use of a top-down network management system (NMS). It therefore follows that the protocol, as part of the Autonomic Networking Infrastructure, should be capable of running in any device that would otherwise be managed by an NMS, and that it can coexist with an NMS and with protocols such as SNMP and NETCONF.

SN7. Specific autonomic features are expected to be implemented by individual ASAs, but the protocol must be general enough to allow them. Some examples follow:

- Dependencies and conflicts: In order to decide upon a configuration for a given device, the device may need information from neighbors. This can be established through the negotiation procedure, or through synchronization if that is sufficient. However, a given item in a neighbor may depend on other information from its own neighbors, which may need another negotiation or synchronization procedure to obtain or decide. Therefore, there are potential dependencies and conflicts among negotiation or synchronization procedures. Resolving dependencies and conflicts is a matter for the individual ASAs involved. To allow this, there need to be clear boundaries and convergence mechanisms for negotiations. Also some mechanisms are needed to avoid loop dependencies or uncontrolled growth in a tree of dependencies. It is the ASA designer's responsibility to avoid or detect looping dependencies or excessive growth of dependency trees. The protocol's role is limited to bilateral signaling between ASAs and the avoidance of loops during bilateral signaling.
- Recovery from faults and identification of faulty devices should be as automatic as possible. The protocol's role is limited to discovery, synchronization, and negotiation. These processes can occur at any time, and an ASA may need to repeat any of these steps when the ASA detects an event such as a negotiation counterpart failing.
- Since a major goal is to minimize human intervention, it is necessary that the network can in effect "think ahead" before changing its parameters. One aspect of this is an ASA that relies on a knowledge base to predict network behavior. This is out of scope for the signaling protocol. However, another aspect is forecasting the effect of a change by a "dry run" negotiation before actually installing the change. Signaling a dry run is therefore a desirable feature of the protocol.

Note that management logging, monitoring, alerts, and tools for intervention are required. However, these can only be features of individual ASAs, not of the protocol itself. Another document [RFC8368] discusses how such agents may be linked into conventional Operations, Administration, and Maintenance (OAM) systems via an Autonomic Control Plane [RFC8994].

SN8. The protocol will be able to deal with a wide variety of technical objectives, covering any type of network parameter. Therefore the protocol will need a flexible and easily extensible format for describing objectives. At a later stage, it may be desirable to adopt an explicit information model. One consideration is whether to adopt an existing information model or to design a new one.

B.3. Specific Technical Requirements

T1. It should be convenient for ASA designers to define new technical objectives and for programmers to express them, without excessive impact on runtime efficiency and footprint. In particular, it should be convenient for ASAs to be implemented independently of each other as user-space programs rather than as kernel code, where such a programming model is possible. The classes of device in which the protocol might run is discussed in [RFC8993].

T2. The protocol should be easily extensible in case the initially defined discovery, synchronization, and negotiation mechanisms prove to be insufficient.

T3. To be a generic platform, the protocol payload format should be independent of the transport protocol or IP version. In particular, it should be able to run over IPv6 or IPv4. However, some functions, such as multicasting on a link, might need to be IP version dependent. By default, IPv6 should be preferred.

T4. The protocol must be able to access off-link counterparts via routable addresses, i.e., must not be restricted to link-local operation.

T5. It must also be possible for an external discovery mechanism to be used, if appropriate for a given technical objective. In other words, GRASP discovery must not be a prerequisite for GRASP negotiation or synchronization.

T6. The protocol must be capable of distinguishing multiple simultaneous operations with one or more peers, especially when wait states occur.

T7. Intent: Although the distribution of Intent is out of scope for this document, the protocol must not by design exclude its use for Intent distribution.

T8. Management monitoring, alerts, and intervention: Devices should be able to report to a monitoring system. Some events must be able to generate operator alerts, and some provision for emergency intervention must be possible (e.g., to freeze synchronization or negotiation in a misbehaving device). These features might not use the signaling protocol itself, but its design should not exclude such use.

T9. Because this protocol may directly cause changes to device configurations and have significant impacts on a running network, all protocol exchanges need to be fully secured against forged messages and man-in-the-middle attacks, and secured as much as reasonably possible against denial-of-service attacks. There must also be an encryption mechanism to resist unwanted monitoring. However, it is not required that the protocol itself provides these security features; it may depend on an existing secure environment.

Appendix C. Capability Analysis of Current Protocols

This appendix discusses various existing protocols with properties related to the requirements described in Appendix B. The purpose is to evaluate whether any existing protocol, or a simple combination of existing protocols, can meet those requirements.

Numerous protocols include some form of discovery, but these all appear to be very specific in their applicability. Service Location Protocol (SLP) [RFC2608] provides service discovery for managed networks, but it requires configuration of its own servers. DNS-Based Service Discovery (DNS-SD) [RFC6763] combined with Multicast DNS (mDNS) [RFC6762] provides service discovery for small networks with a single link layer. [RFC7558] aims to extend this to larger autonomous networks, but this is not yet standardized. However, both SLP and DNS-SD appear to target primarily application-layer services, not the layer 2 and 3 objectives relevant to basic network configuration. Both SLP and DNS-SD are text-based protocols.

Simple Network Management Protocol (SNMP) [RFC3416] uses a command/response model not well suited for peer negotiation. NETCONF [RFC6241] uses an RPC model that does allow positive or negative responses from the target system, but this is still not adequate for negotiation.

There are various existing protocols that have elementary negotiation abilities, such as Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [RFC8415], Neighbor Discovery (ND) [RFC4861], Port Control Protocol (PCP) [RFC6887], Remote Authentication Dial-In User Service (RADIUS) [RFC2865], Diameter [RFC6733], etc. Most of them are configuration or management protocols. However, they either provide only a simple request/response model in a master/slave context or very limited negotiation abilities.

There are some signaling protocols with an element of negotiation. For example, Resource ReSerVation Protocol (RSVP) [RFC2205] was designed for negotiating quality-of-service parameters along the path of a unicast or multicast flow. RSVP is a very specialized protocol aimed at end-to-end flows. A more generic design is General Internet Signalling Transport (GIST) [RFC5971]; however, it tries to solve many problems, making it complex, and is also aimed at per-flow signaling across many hops rather than at device-to-device signaling. However, we cannot completely exclude extended RSVP or GIST as a synchronization and negotiation protocol. They do not appear to be directly usable for peer discovery.

RESTCONF [RFC8040] is a protocol intended to convey NETCONF information expressed in the YANG language via HTTP, including the ability to transit HTML intermediaries. While this is a powerful approach in the context of centralized configuration of a complex network, it is not well adapted to efficient interactive negotiation between peer devices, especially simple ones that might not include YANG processing already.

The Distributed Node Consensus Protocol (DNCP) [RFC7787] is defined as a generic form of a state synchronization protocol, with a proposed usage profile being the Home Networking Control Protocol (HNCP) [RFC7788] for configuring Homenet routers. A specific application of DNCP for Autonomic Networking was proposed in [ADNCP]. According to [RFC7787]:

> DNCP is designed to provide a way for each participating node to publish a set of TLV (Type-Length-Value) tuples (at most 64 KB) and to provide a shared and common view about the data published...
>
> DNCP is most suitable for data that changes only infrequently...
>
> If constant rapid state changes are needed, the preferable choice is to use an additional point-to-point channel...

Specific features of DNCP include:

- Every participating node has a unique node identifier.
- DNCP messages are encoded as a sequence of TLV objects and sent over unicast UDP or TCP, with or without (D)TLS security.
- Multicast is used only for discovery of DNCP neighbors when lower security is acceptable.
- Synchronization of state is maintained by a flooding process using the Trickle algorithm. There is no bilateral synchronization or negotiation capability.
- The HNCP profile of DNCP is designed to operate between directly connected neighbors on a shared link using UDP and link-local IPv6 addresses.

DNCP does not meet the needs of a general negotiation protocol because it is designed specifically for flooding synchronization. Also, in its HNCP profile, it is limited to link-local messages and to IPv6. However, at the minimum, it is a very interesting test case for this style of interaction between devices without needing a central authority, and it is a proven method of network-wide state synchronization by flooding.

The Server Cache Synchronization Protocol (SCSP) [RFC2334] also describes a method for cache synchronization and cache replication among a group of nodes.

A proposal was made some years ago for an IP based Generic Control Protocol (IGCP) [IGCP]. This was aimed at information exchange and negotiation but not directly at peer discovery. However, it has many points in common with the present work.

None of the above solutions appears to completely meet the needs of generic discovery, state synchronization, and negotiation in a single solution. Many of the protocols assume that they are working in a traditional top-down or north-south scenario, rather than a fluid peer-to-peer scenario. Most of them are specialized in one way or another. As a result, we have not identified a combination of existing protocols that meets the requirements in Appendix B. Also, we have not identified a path by which one of the existing protocols could be extended to meet the requirements.

Acknowledgments

A major contribution to the original draft version of this document was made by Sheng Jiang, and significant contributions were made by Toerless Eckert. Significant early review inputs were received from Joel Halpern, Barry Leiba, Charles E. Perkins, and Michael Richardson. William Atwood provided important assistance in debugging a prototype implementation.

Valuable comments were received from Michael Behringer, Jéferson Campos Nobre, Laurent Ciavaglia, Zongpeng Du, Yu Fu, Joel Jaeggli, Zhenbin Li, Dimitri Papadimitriou, Pierre Peloso, Reshad Rahman, Markus Stenberg, Martin Stiemerling, Rene Struik, Martin Thomson, Dacheng Zhang, and participants in the Network Management Research Group, the ANIMA Working Group, and the IESG.

Authors' Addresses

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Email: cabo@tzi.org

Brian Carpenter (EDITOR)
School of Computer Science
University of Auckland
PB 92019
Auckland 1142
New Zealand
Email: brian.e.carpenter@gmail.com

Bing Liu (EDITOR)
Huawei Technologies Co., Ltd
Q14, Huawei Campus
Hai-Dian District
No.156 Beiqing Road
Beijing
100095
China
Email: leo.liubing@huawei.com