Network Working Group
Internet-Draft
Intended status: Informational
Expires: 26 January 2021

B. E. Carpenter
Univ. of Auckland
L. Ciavaglia
Nokia
S. Jiang
Huawei Technologies Co., Ltd
P. Peloso
Nokia

25 July 2020

# Guidelines for Autonomic Service Agents
draft-carpenter-anima-asa-guidelines-09

## Abstract

This document proposes guidelines for the design of Autonomic Service Agents for autonomic networks, as a contribution to describing an autonomic ecosystem. It is based on the Autonomic Network Infrastructure outlined in the ANIMA reference model, using the Autonomic Control Plane and the Generic Autonomic Signaling Protocol.

## Status of This Memo



## Copyright Notice

Table of Contents



## 1. Introduction

This document proposes guidelines for the design of Autonomic Service Agents (ASAs) in the context of an Autonomic Network (AN) based on the Autonomic Network Infrastructure (ANI) outlined in the ANIMA reference model [I-D.ietf-anima-reference-model]. This infrastructure makes use of the Autonomic Control Plane (ACP) [I-D.ietf-anima-autonomic-control-plane] and the Generic Autonomic Signaling Protocol (GRASP) [I-D.ietf-anima-grasp]. This document is a contribution to the description of an autonomic ecosystem, recognizing that a deployable autonomic network needs more than just

ACP and GRASP implementations. It must achieve management goals that a Network Operations Center (NOC) cannot achieve manually, including at least a library of ASAs and corresponding GRASP objective definitions. There must also be tools to deploy and oversee ASAs, and integration with existing operational mechanisms [RFC8368]. However, this document focuses on the design of ASAs, with some reference to implementation and operational aspects.

There is a considerable literature about autonomic agents with a variety of proposals about how they should be characterized. Some examples are [DeMola06], [Huebscher08], [Movahedi12] and [GANA13]. However, for the present document, the basic definitions and goals for autonomic networking given in [RFC7575] apply . According to RFC 7575, an Autonomic Service Agent is "An agent implemented on an autonomic node that implements an autonomic function, either in part (in the case of a distributed function) or whole."

ASAs must be distinguished from other forms of software component. They are components of network or service management; they do not in themselves provide services. For example, the services envisaged for network function virtualisation [RFC8568] or for service function chaining [RFC7665] might be managed by an ASA rather than by traditional configuration tools.

The reference model [I-D.ietf-anima-reference-model] expands this by adding that an ASA is "a process that makes use of the features provided by the ANI to achieve its own goals, usually including interaction with other ASAs via the GRASP protocol [I-D.ietf-anima-grasp] or otherwise. Of course it also interacts with the specific targets of its function, using any suitable mechanism. Unless its function is very simple, the ASA will need to handle overlapping asynchronous operations. This will require either a multi-threaded implementation, or a logically equivalent event loop structure. It may therefore be a quite complex piece of software in its own right, forming part of the application layer above the ANI."

There will certainly be very simple ASAs that manage a single objective in a straightforward way and do not need asynchronous operations. In such a case, many aspects of the current document do not apply. However, in general a basic property of an ASA is that it is a relatively complex software component that will in many cases control and monitor simpler entities in the same host or elsewhere. For example, a device controller that manages tens or hundreds of simple devices might contain a single ASA.

The remainder of this document offers guidance on the design of such ASAs.

## 2. Logical Structure of an Autonomic Service Agent

As mentioned above, all but the simplest ASAs will need to suport asynchronous operations. Not all programming environments explicitly support multi-threading. In that case, an 'event loop' style of implementation should be adopted, in which case each thread would be implemented as an event handler called in turn by the main loop. For this, the GRASP API (Section 3.3) must provide non-blocking calls. If necessary, the GRASP session identifier will be used to distinguish simultaneous operations.

A typical ASA will have a main thread that performs various initial housekeeping actions such as:

* Obtain authorization credentials.

* Register the ASA with GRASP.

* Acquire relevant policy parameters.

* Define data structures for relevant GRASP objectives.

* Register with GRASP those objectives that it will actively manage.

* Launch a self-monitoring thread.

* Enter its main loop.

The logic of the main loop will depend on the details of the autonomic function concerned. Whenever asynchronous operations are required, extra threads will be launched, or events added to the event loop. Examples include:

* Repeatedly flood an objective to the AN, so that any ASA can receive the objective's latest value.

* Accept incoming synchronization requests for an objective managed by this ASA.

* Accept incoming negotiation requests for an objective managed by this ASA, and then conduct the resulting negotiation with the counterpart ASA.

* Manage subsidiary non-autonomic devices directly.

These threads or events should all either exit after their job is done, or enter a wait state for new work, to avoid blocking others unnecessarily.

According to the degree of parallelism needed by the application, some of these threads or events might be launched in multiple instances. In particular, if negotiation sessions with other ASAs are expected to be long or to involve wait states, the ASA designer might allow for multiple simultaneous negotiating threads, with appropriate use of queues and locks to maintain consistency.

The main loop itself could act as the initiator of synchronization requests or negotiation requests, when the ASA needs data or resources from other ASAs. In particular, the main loop should watch for changes in policy parameters that affect its operation. It should also do whatever is required to avoid unnecessary resource consumption, such as including an arbitrary wait time in each cycle of the main loop.

The self-monitoring thread is of considerable importance. Autonomic service agents must never fail. To a large extent this depends on careful coding and testing, with no unhandled error returns or exceptions, but if there is nevertheless some sort of failure, the self-monitoring thread should detect it, fix it if possible, and in the worst case restart the entire ASA.

Appendix B presents some example logic flows in informal pseudocode.

# 3. Interaction with the Autonomic Networking Infrastructure

## 3.1. Interaction with the security mechanisms

An ASA by definition runs in an autonomic node. Before any normal ASAs are started, such nodes must be bootstrapped into the autonomic network's secure key infrastructure in accordance with [I-D.ietf-anima-bootstrapping-keyinfra]. This key infrastructure will be used to secure the ACP (next section) and may be used by ASAs to set up additional secure interactions with their peers, if needed.

Note that the secure bootstrap process itself may include special-purpose ASAs that run in a constrained insecure mode.

## 3.2. Interaction with the Autonomic Control Plane

In a normal autonomic network, ASAs will run as users of the ACP, which will provide a fully secured network environment for all communication with other ASAs, in most cases mediated by GRASP (next section).

Note that the ACP formation process itself may include special-purpose ASAs that run in a constrained insecure mode.

### 3.3. Interaction with GRASP and its API

GRASP [I-D.ietf-anima-grasp] is expected to run as a separate process with its API [I-D.ietf-anima-grasp-api] available in user space. Thus ASAs may operate without special privilege, unless they need it for other reasons. The ASA's view of GRASP is built around GRASP objectives (Section 5), 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 serialise it for transmission in CBOR [RFC7049], which is no restriction at all in practice.

The GRASP API should offer the following features:

* Registration functions, so that an ASA can register itself and the objectives that it manages.

* A discovery function, by which an ASA can discover other ASAs supporting a given objective.

* A negotiation request function, by which an ASA can start negotiation of an objective with a counterpart ASA. With this, there is a corresponding listening function for an ASA that wishes to respond to negotiation requests, and a set of functions to support negotiating steps.

* A synchronization function, by which an ASA can request the current value of an objective from a counterpart ASA. With this, there is a corresponding listening function for an ASA that wishes to respond to synchronization requests.

* A flood function, by which an ASA can cause the current value of an objective to be flooded throughout the AN so that any ASA can receive it.

For further details and some additional housekeeping functions, see [I-D.ietf-anima-grasp-api].

This API is intended to support the various interactions expected between most ASAs, such as the interactions outlined in Section 2. However, if ASAs require additional communication between themselves, they can do so using any desired protocol. One option is to use GRASP discovery and synchronization as a rendez-vous mechanism between two ASAs, passing communication parameters such as a TCP port number via GRASP. As noted above, either the ACP or in special cases the autonomic key infrastructure will be used to secure such communications.

### 3.4. Interaction with policy mechanism

At the time of writing, the policy (or "Intent") mechanism for the ANI is undefined and is regarded as a research topic. It is expected to operate by an information distribution mechanism (e.g. [I-D.liu-anima-grasp-distribution]) that can reach all autonomic nodes, and therefore every ASA. However, each ASA must be capable of operating "out of the box" in the absence of locally defined policy, so every ASA implementation must include carefully chosen default values and settings for all policy parameters.

## 4. Interaction with Non-Autonomic Components

An ASA, to have any external effects, must also interact with non-autonomic components of the node where it is installed. For example, an ASA whose purpose is to manage a resource must interact with that resource. An ASA whose purpose is to manage an entity that is already managed by local software must interact with that software. For example, if such management is performed by NETCONF [RFC6241], the ASA must interact directly with the NETCONF server in the same node. This is stating the obvious, and the details are specific to each case, but it has an important security implication. The ASA might act as a loophole by which the managed entity could penetrate the security boundary of the ANI. The ASA must be designed to avoid such loopholes, and should if possible operate in an unprivileged mode.

In an environment where systems are virtualized and specialized using techniques such as network function virtualization or network slicing, there will be a design choice whether ASAs are deployed once per physical node or once per virtual context. A related issue is whether the ANI as a whole is deployed once on a physical network, or whether several virtual ANIs are deployed. This aspect needs to be considered by the ASA designer.

## 5. Design of GRASP Objectives

The general rules for the format of GRASP Objective options, their names, and IANA registration are given in [I-D.ietf-anima-grasp]. Additionally that document discusses various general considerations for the design of objectives, which are not repeated here. However, we emphasize that the GRASP protocol does not provide transactional integrity. In other words, if an ASA is capable of overlapping several negotiations for a given objective, then the ASA itself must use suitable locking techniques to avoid interference between these negotiations. For example, if an ASA is allocating part of a shared resource to other ASAs, it needs to ensure that the same part of the resource is not allocated twice. This might impact the design of the objective as well as the logic flow of the ASA.

In particular, if 'dry run' mode is defined for the objective, its specification, and every implementation, must consider what state needs to be saved following a dry run negotiation, such that a subsequent live negotiation can be expected to succeed. It must be clear how long this state is kept, and what happens if the live negotiation occurs after this state is deleted. An ASA that requests a dry run negotiation must take account of the possibility that a successful dry run is followed by a failed live negotiation. Because of these complexities, the dry run mechanism should only be supported by objectives and ASAs where there is a significant benefit from it.

The actual value field of an objective is limited by the GRASP protocol definition to any data structure that can be expressed in Concise Binary Object Representation (CBOR) [RFC7049]. For some objectives, a single data item will suffice; for example an integer, a floating point number or a UTF-8 string. For more complex cases, a simple tuple structure such as [item1, item2, item3] could be used. Nothing prevents using other formats such as JSON, but this requires the ASA to be capable of parsing and generating JSON. The formats acceptable by the GRASP API will limit the options in practice. A fallback solution is for the API to accept and deliver the value field in raw CBOR, with the ASA itself encoding and decoding it via a CBOR library.

Note that a mapping from YANG to CBOR is defined by [I-D.ietf-core-yang-cbor]. Subject to the size limit defined for GRASP messages, nothing prevents objectives using YANG in this way.

## 6. Life Cycle

Autonomic functions could be permanent, in the sense that ASAs are shipped as part of a product and persist throughout the product's life. However, a more likely situation is that ASAs need to be installed or updated dynamically, because of new requirements or bugs. Because continuity of service is fundamental to autonomic networking, the process of seamlessly replacing a running instance of an ASA with a new version needs to be part of the ASA's design.

The implication of service continuity on the design of ASAs can be illustrated along the three main phases of the ASA life-cycle, namely Installation, Instantiation and Operation.

```
                  +--------------+
Undeployed ------>|              |------> Undeployed
                  |   Installed  |
              +-->|              |---+
      Mandate |   +--------------+   | Receives a
   is revoked |   +--------------+   |  Mandate
              +---|              |<--+
                  | Instantiated |
              +-->|              |---+
          set |   +--------------+   | set
         down |   +--------------+   | up
              +---|              |<--+
                  |  Operational |
                  |              |
                  +--------------+
```

Figure 1: Life cycle of an Autonomic Service Agent

### 6.1. Installation phase

Before being able to instantiate and run ASAs, the operator must first provision the infrastructure with the sets of ASA software corresponding to its needs and objectives. The provisioning of the infrastructure is realized in the installation phase and consists in installing (or checking the availability of) the pieces of software of the different ASA classes in a set of Installation Hosts.

There are 3 properties applicable to the installation of ASAs:

The dynamic installation property  allows installing an ASA on
   demand, on any hosts compatible with the ASA.

The decoupling property allows controlling resources of a NE from a remote ASA, i.e. an ASA installed on a host machine different from the resources' NE.

The multiplicity property allows controlling multiple sets of resources from a single ASA.

These three properties are very important in the context of the installation phase as their variations condition how the ASA class could be installed on the infrastructure.

#### 6.1.1. Installation phase inputs and outputs

Inputs are:

[ASA class of type_x] that specifies which classes ASAs to install,

[Installation_target_Infrastructure] that specifies the candidate Installation Hosts,

[ASA class placement function, e.g. under which criteria/ constraints as defined by the operator] that specifies how the installation phase shall meet the operator's needs and objectives for the provision of the infrastructure. In the coupled mode, the placement function is not necessary, whereas in the decoupled mode, the placement function is mandatory, even though it can be as simple as an explicit list of Installation hosts.

The main output of the installation phase is an up-to-date directory of installed ASAs which corresponds to [list of ASA classes] installed on [list of installation Hosts]. This output is also useful for the coordination function and corresponds to the static interaction map (see next section).

The condition to validate in order to pass to next phase is to ensure that [list of ASA classes] are well installed on [list of installation Hosts]. The state of the ASA at the end of the installation phase is: installed. (not instantiated). The following commands or messages are foreseen: install(list of ASA classes, Installation_target_Infrastructure, ASA class placement function), and un-install (list of ASA classes).

### 6.2. Instantiation phase

Once the ASAs are installed on the appropriate hosts in the network, these ASA may start to operate. From the operator viewpoint, an operating ASA means the ASA manages the network resources as per the objectives given. At the ASA local level, operating means executing their control loop/algorithm.

But right before that, there are two things to take into consideration. First, there is a difference between 1. having a piece of code available to run on a host and 2. having an agent based on this piece of code running inside the host. Second, in a coupled case, determining which resources are controlled by an ASA is straightforward (the determination is embedded), in a decoupled mode determining this is a bit more complex (hence a starting agent will have to either discover or be taught it).

The instantiation phase of an ASA covers both these aspects: starting the agent piece of code (when this does not start automatically) and determining which resources have to be controlled (when this is not obvious).

#### 6.2.1. Operator's goal

Through this phase, the operator wants to control its autonomic network in two things:

1 determine the scope of autonomic functions by instructing which of the network resources have to be managed by which autonomic function (and more precisely which class e.g. 1. version X or version Y or 2. provider A or provider B),

2 determine how the autonomic functions are organized by instructing which ASAs have to interact with which other ASAs (or more precisely which set of network resources have to be handled as an autonomous group by their managing ASAs).

Additionally in this phase, the operator may want to set objectives to autonomic functions, by configuring the ASAs technical objectives.

The operator's goal can be summarized in an instruction to the ANIMA ecosystem matching the following pattern:

```
[ASA of type_x instances] ready to control
[Instantiation_target_Infrastructure] with
[Instantiation_target_parameters]
```

#### 6.2.2. Instantiation phase inputs and outputs

Inputs are:

[ASA of type_x instances] that specifies which are the ASAs to be targeted (and more precisely which class e.g. 1. version X or version Y or 2. provider A or provider B),

[Instantiation_target_Infrastructure] that specifies which are the resources to be managed by the autonomic function, this can be the whole network or a subset of it like a domain a technology segment or even a specific list of resources,

[Instantiation_target_parameters] that specifies which are the technical objectives to be set to ASAs (e.g. an optimization target)

Outputs are:

[Set of ASAs - Resources relations] describing which resources are managed by which ASA instances, this is not a formal message, but a resulting configuration of a set of ASAs,

#### 6.2.3. Instantiation phase requirements

The instructions described in section 4.2 could be either:

sent to a targeted ASA In which case, the receiving Agent will have to manage the specified list of [Instantiation_target_Infrastructure], with the [Instantiation_target_parameters].

broadcast to all ASAs In which case, the ASAs would collectively determine from the list which Agent(s) would handle which [Instantiation_target_Infrastructure], with the [Instantiation_target_parameters].

This set of instructions can be materialized through a message that is named an Instance Mandate (description TBD).

The conclusion of this instantiation phase is a ready to operate ASA (or interacting set of ASAs), then this (or those) ASA(s) can describe themselves by depicting which are the resources they manage and what this means in terms of metrics being monitored and in terms of actions that can be executed (like modifying the parameters values). A message conveying such a self description is named an Instance Manifest (description TBD).

Though the operator may well use such a self-description "per se", the final goal of such a description is to be shared with other ANIMA entities like:

* the coordination entities (see [I-D.ciavaglia-anima-coordination])

* collaborative entities in the purpose of establishing knowledge exchanges (some ASAs may produce knowledge or even monitor metrics that other ASAs cannot make by themselves why those would be useful for their execution)

### 6.3. Operation phase

Note: This section is to be further developed in future revisions of the document, especially the implications on the design of ASAs.

During the Operation phase, the operator can:

Activate/Deactivate ASA: meaning enabling those to execute their autonomic loop or not.

Modify ASAs targets: meaning setting them different objectives.

Modify ASAs managed resources: by updating the instance mandate which would specify different set of resources to manage (only applicable to decouples ASAs).

During the Operation phase, running ASAs can interact the one with the other:

in order to exchange knowledge (e.g. an ASA providing traffic predictions to load balancing ASA)

in order to collaboratively reach an objective (e.g. ASAs pertaining to the same autonomic function targeted to manage a network domain, these ASA will collaborate - in the case of a load balancing one, by modifying the links metrics according to the neighboring resources loads)

During the Operation phase, running ASAs are expected to apply coordination schemes

then execute their control loop under coordination supervision/ instructions

The ASA life-cycle is discussed in more detail in "A Day in the Life of an Autonomic Function" [I-D.peloso-anima-autonomic-function].

## 7. Coordination between Autonomic Functions

Some autonomic functions will be completely independent of each other. However, others are at risk of interfering with each other - for example, two different optimization functions might both attempt to modify the same underlying parameter in different ways. In a complete system, a method is needed of identifying ASAs that might interfere with each other and coordinating their actions when necessary. This issue is considered in "Autonomic Functions Coordination" [I-D.ciavaglia-anima-coordination].

## 8. Coordination with Traditional Management Functions

Some ASAs will have functions that overlap with existing configuration tools and network management mechanisms such as command line interfaces, DHCP, DHCPv6, SNMP, NETCONF, RESTCONF and YANG-based solutions. Each ASA designer will need to consider this issue and how to avoid clashes and inconsistencies. Some specific considerations for interaction with OAM tools are given in [RFC8368]. As another example, [I-D.ietf-anima-prefix-management] describes how autonomic management of IPv6 prefixes can interact with prefix delegation via DHCPv6. The description of a GRASP objective and of an ASA using it should include a discussion of any such interactions.

A related aspect is that management functions often include a data model, quite likely to be expressed in a formal notation such as YANG. This aspect should not be an afterthought in the design of an ASA. To the contrary, the design of the ASA and of its GRASP objectives should match the data model; as noted above, YANG serialized as CBOR may be used directly as the value of a GRASP objective.

## 9. Robustness

It is of great importance that all components of an autonomic system are highly robust. In principle they must never fail. This section lists various aspects of robustness that ASA designers should consider.

1. If despite all precautions, an ASA does encounter a fatal error, it should in any case restart automatically and try again. To mitigate a hard loop in case of persistent failure, a suitable pause should be inserted before such a restart. The length of the pause depends on the use case.

2. If a newly received or calculated value for a parameter falls out of bounds, the corresponding parameter should be either left unchanged or restored to a safe value.

3. If a GRASP synchronization or negotiation session fails for any reason, it may be repeated after a suitable pause. The length of the pause depends on the use case.

4. If a session fails repeatedly, the ASA should consider that its peer has failed, and cause GRASP to flush its discovery cache and repeat peer discovery.

5. In any case, it may be prudent to repeat discovery periodically, depending on the use case.

6. Any received GRASP message should be checked. If it is wrongly formatted, it should be ignored. Within a unicast session, an Invalid message (M_INVALID) may be sent. This function may be provided by the GRASP implementation itself.

7. Any received GRASP objective should be checked. If it is wrongly formatted, it should be ignored. Within a negotiation session, a Negotiation End message (M_END) with a Decline option (O_DECLINE) should be sent. An ASA may log such events for diagnostic purposes.

8. If an ASA receives either an Invalid message (M_INVALID) or a Negotiation End message (M_END) with a Decline option (O_DECLINE), one possible reason is that the peer ASA does not support a new feature of either GRASP or of the objective in question. In such a case the ASA may choose to repeat the operation concerned without using that new feature.

9. All other possible exceptions should be handled in an orderly way. There should be no such thing as an unhandled exception (but see point 1 above).

## 10. Security Considerations

ASAs are intended to run in an environment that is protected by the Autonomic Control Plane [I-D.ietf-anima-autonomic-control-plane], admission to which depends on an initial secure bootstrap process [I-D.ietf-anima-bootstrapping-keyinfra]. In some deployments, a secure partition of the link layer might be used instead [I-D.carpenter-anima-l2acp-scenarios]. However, this does not relieve ASAs of responsibility for security. In particular, when ASAs configure or manage network elements outside the ACP, they must use secure techniques and carefully validate any incoming information. As noted above, this will apply in particular when an ASA interacts with a management component such as a NETCONF server.

As appropriate to their specific functions, ASAs should take account of relevant privacy considerations [RFC6973].

Authorization of ASAs is a subject for future study. At present, ASAs are trusted by virtue of being installed on a node that has successfully joined the ACP. In the general case, a node may have mutltiple roles and a role may use multiple ASAs, each using multiple GRASP objectives. Additional mechanisms for the authorization of nodes and ASAs to manipulate specific GRASP objectives could be designed.

## 11. IANA Considerations

This document makes no request of the IANA.

## 12. Acknowledgements

Useful comments were received from Michael Behringer Toerless Eckert, Alex Galis, Bing Liu, Michael Richardson, and other members of the ANIMA WG.

## 13. References

### 13.1. Normative References

[I-D.ietf-anima-autonomic-control-plane]
Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic Control Plane (ACP)", Work in Progress, Internet-Draft, draft-ietf-anima-autonomic-control-plane-27, 2 July 2020, <https://tools.ietf.org/html/draft-ietf-anima-autonomic-control-plane-27>.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-bootstrapping-keyinfra-41, 8 April 2020, <https://tools.ietf.org/html/draft-ietf-anima-bootstrapping-keyinfra-41>.

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", Work in Progress, Internet-Draft, draft-ietf-anima-grasp-15, 13 July 2017, <https://tools.ietf.org/html/draft-ietf-anima-grasp-15>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

## 13.2. Informative References

[DeMola06] De Mola, F. and R. Quitadamo, "An Agent Model for Future Autonomic Communications", Proceedings of the 7th WOA 2006 Workshop From Objects to Agents 51-59, September 2006.

[GANA13] "Autonomic network engineering for the self-managing Future Internet (AFI): GANA Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management.", April 2013, <http://www.etsi.org/deliver/etsi_gs/AFI/001_099/002/01.01.01_60/gs_afi002v010101p.pdf>.

[Huebscher08] Huebscher, M. C. and J. A. McCann, "A survey of autonomic computing--degrees, models, and applications", ACM Computing Surveys (CSUR) Volume 40 Issue 3 DOI: 10.1145/1380584.1380585, August 2008.

[I-D.carpenter-anima-l2acp-scenarios] Carpenter, B. and B. Liu, "Scenarios and Requirements for Layer 2 Autonomic Control Planes", Work in Progress, Internet-Draft, draft-carpenter-anima-l2acp-scenarios-02, 8 April 2020, <https://tools.ietf.org/html/draft-carpenter-anima-l2acp-scenarios-02>.

[I-D.ciavaglia-anima-coordination] Ciavaglia, L. and P. Peloso, "Autonomic Functions Coordination", Work in Progress, Internet-Draft, draft-ciavaglia-anima-coordination-01, 21 March 2016, <https://tools.ietf.org/html/draft-ciavaglia-anima-coordination-01>.

[I-D.ietf-anima-grasp-api] Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)", Work in Progress, Internet-Draft, draft-ietf-anima-grasp-api-06, 12 June 2020, <https://tools.ietf.org/html/draft-ietf-anima-grasp-api-06>.

[I-D.ietf-anima-prefix-management] Jiang, S., Du, Z., Carpenter, B., and Q. Sun, "Autonomic IPv6 Edge Prefix Management in Large-scale Networks", Work in Progress, Internet-Draft, draft-ietf-anima-prefix-management-07, 18 December 2017, <https://tools.ietf.org/html/draft-ietf-anima-prefix-management-07>.

[I-D.ietf-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", Work in Progress, Internet-Draft, draft-ietf-anima-reference-model-10, 22 November 2018, <https://tools.ietf.org/html/draft-ietf-anima-reference-model-10>.

[I-D.ietf-core-yang-cbor]
Veillette, M., Petrov, I., and A. Pelov, "CBOR Encoding of Data Modeled with YANG", Work in Progress, Internet-Draft, draft-ietf-core-yang-cbor-13, 4 July 2020, <https://tools.ietf.org/html/draft-ietf-core-yang-cbor-13>.

[I-D.liu-anima-grasp-distribution]
Liu, B., Xiao, X., Hecker, A., Jiang, S., and Z. Despotovic, "Information Distribution in Autonomic Networking", Work in Progress, Internet-Draft, draft-liu-anima-grasp-distribution-13, 12 December 2019, <https://tools.ietf.org/html/draft-liu-anima-grasp-distribution-13>.

[I-D.peloso-anima-autonomic-function]
Pierre, P. and L. Ciavaglia, "A Day in the Life of an Autonomic Function", Work in Progress, Internet-Draft, draft-peloso-anima-autonomic-function-01, 21 March 2016, <https://tools.ietf.org/html/draft-peloso-anima-autonomic-function-01>.

[Movahedi12]
Movahedi, Z., Ayari, M., Langar, R., and G. Pujolle, "A Survey of Autonomic Network Architectures and Evaluation Criteria", IEEE Communications Surveys & Tutorials Volume: 14 , Issue: 2 DOI: 10.1109/SURV.2011.042711.00078, Page(s): 464 - 490, 2012.

[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>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[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>.

[RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

[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>.

[RFC8568] Bernardos, CJ., Rahman, A., Zuniga, JC., Contreras, LM., Aranda, P., and P. Lynch, "Network Virtualization Research Challenges", RFC 8568, DOI 10.17487/RFC8568, April 2019, <https://www.rfc-editor.org/info/rfc8568>.

## Appendix A. Change log [RFC Editor: Please remove]

draft-carpenter-anima-asa-guidelines-09, 2020-07-25:

* Additional text on future authorization.
* Editorial fixes

draft-carpenter-anima-asa-guidelines-08, 2020-01-10:

* Introduced notion of autonomic ecosystem.
* Minor technical clarifications.
* Converted to v3 format.

draft-carpenter-anima-asa-guidelines-07, 2019-07-17:

* Improved explanation of threading vs event-loop
* Other editorial improvements.

draft-carpenter-anima-asa-guidelines-06, 2018-01-07:

* Expanded and improved example logic flow.
* Editorial corrections.

draft-carpenter-anima-asa-guidelines-05, 2018-06-30:

* Added section on relationshp with non-autonomic components.
* Editorial corrections.

draft-carpenter-anima-asa-guidelines-04, 2018-03-03:

* Added note about simple ASAs.
* Added note about NFV/SFC services.
* Improved text about threading v event loop model
* Added section about coordination with traditional tools.
* Added appendix with example logic flow.

draft-carpenter-anima-asa-guidelines-03, 2017-10-25:

* Added details on life cycle.
* Added details on robustness.
* Added co-authors.

draft-carpenter-anima-asa-guidelines-02, 2017-07-01:

* Expanded description of event-loop case.
* Added note about 'dry run' mode.

draft-carpenter-anima-asa-guidelines-01, 2017-01-06:

* More sections filled in.

draft-carpenter-anima-asa-guidelines-00, 2016-09-30:

* Initial version

## Appendix B. Example Logic Flows

This appendix describes generic logic flows for an Autonomic Service Agent (ASA) for resource management. Note that these are illustrative examples, and in no sense requirements. As long as the rules of GRASP are followed, a real implementation could be different. The reader is assumed to be familiar with GRASP [I-D.ietf-anima-grasp] and its conceptual API [I-D.ietf-anima-grasp-api].

A complete autonomic function for a resource would consist of a number of instances of the ASA placed at relevant points in a network. Specific details will of course depend on the resource concerned. One example is IP address prefix management, as specified in [I-D.ietf-anima-prefix-management]. In this case, an instance of the ASA would exist in each delegating router.

An underlying assumption is that there is an initial source of the resource in question, referred to here as an origin ASA. The other ASAs, known as delegators, obtain supplies of the resource from the origin, and then delegate quantities of the resource to consumers that request it, and recover it when no longer needed.

Another assumption is there is a set of network wide policy parameters, which the origin will provide to the delegators. These parameters will control how the delegators decide how much resource to provide to consumers. Thus the ASA logic has two operating modes: origin and delegator. When running as an origin, it starts by obtaining a quantity of the resource from the NOC, and it acts as a source of policy parameters, via both GRASP flooding and GRASP synchronization. (In some scenarios, flooding or synchronization alone might be sufficient, but this example includes both.)

When running as a delegator, it starts with an empty resource pool, it acquires the policy parameters by GRASP synchronization, and it delegates quantities of the resource to consumers that request it. Both as an origin and as a delegator, when its pool is low it seeks quantities of the resource by requesting GRASP negotiation with peer ASAs. When its pool is sufficient, it hands out resource to peer ASAs in response to negotiation requests. Thus, over time, the initial resource pool held by the origin will be shared among all the delegators according to demand.

In theory a network could include any number of origins and any number of delegators, with the only condition being that each origin's initial resource pool is unique. A realistic scenario is to have exactly one origin and as many delegators as you like. A scenario with no origin is useless.

An implementation requirement is that resource pools are kept in stable storage. Otherwise, if a delegator exits for any reason, all the resources it has obtained or delegated are lost. If an origin exits, its entire spare pool is lost. The logic for using stable storage and for crash recovery is not included in the pseudocode below.

The description below does not implement GRASP's 'dry run' function. That would require temporarily marking any resource handed out in a dry run negotiation as reserved, until either the peer obtains it in a live run, or a suitable timeout expires.

The main data structures used in each instance of the ASA are:

* The resource_pool, for example an ordered list of available resources. Depending on the nature of the resource, units of resource are split when appropriate, and a background garbage collector recombines split resources if they are returned to the pool.

* The delegated_list, where a delegator stores the resources it has given to consumers routers.

Possible main logic flows are below, using a threaded implementation model. The transformation to an event loop model should be apparent - each thread would correspond to one event in the event loop.

The GRASP objectives are as follows:

* ["EX1.Resource", flags, loop_count, value] where the value depends on the resource concerned, but will typically include its size and identification.

* ["EX1.Params", flags, loop_count, value] where the value will be, for example, a JSON object defining the applicable parameters.

In the outline logic flows below, these objectives are represented simply by their names.

```
<CODE BEGINS>

MAIN PROGRAM:

Create empty resource_pool (and an associated lock)
Create empty delegated_list
Determine whether to act as origin
if origin:
    Obtain initial resource_pool contents from NOC
    Obtain value of EX1.Params from NOC
Register ASA with GRASP
Register GRASP objectives EX1.Resource and EX1.Params
if origin:
    Start FLOODER thread to flood EX1.Params
    Start SYNCHRONIZER listener for EX1.Params
Start MAIN_NEGOTIATOR thread for EX1.Resource
if not origin:
    Obtain value of EX1.Params from GRASP flood or synchronization
    Start DELEGATOR thread
Start GARBAGE_COLLECTOR thread
do forever:
    good_peer = none
    if resource_pool is low:
        Calculate amount A of resource needed
        Discover peers using GRASP M_DISCOVER / M_RESPONSE
        if good_peer in peers:
            peer = good_peer
        else:
            peer =  #any choice among peers
            grasp.request_negotiate("EX1.Resource", peer)
            i.e., send M_REQ_NEG
            Wait for response (M_NEGOTIATE, M_END or M_WAIT)
            if OK:
                if offered amount of resource sufficient:
                    Send M_END + O_ACCEPT #negotiation succeeded
                    Add resource to pool
                    good_peer = peer
                else:
                    Send M_END + O_DECLINE #negotiation failed
    sleep() #sleep time depends on application scenario

MAIN_NEGOTIATOR thread:

do forever:
    grasp.listen_negotiate("EX1.Resource")
    i.e., wait for M_REQ_NEG
    Start a separate new NEGOTIATOR thread for requested amount A
```

```
NEGOTIATOR thread:

Request resource amount A from resource_pool
if not OK:
    while not OK and A > Amin:
        A = A-1
        Request resource amount A from resource_pool
if OK:
    Offer resource amount A to peer by GRASP M_NEGOTIATE
    if received M_END + O_ACCEPT:
        #negotiation succeeded
    elif received M_END + O_DECLINE or other error:
        #negotiation failed
else:
    Send M_END + O_DECLINE #negotiation failed

DELEGATOR thread:

do forever:
    Wait for request or release for resource amount A
    if request:
        Get resource amount A from resource_pool
        if OK:
            Delegate resource to consumer
            Record in delegated_list
        else:
            Signal failure to consumer
            Signal main thread that resource_pool is low
    else:
        Delete resource from delegated_list
        Return resource amount A to resource_pool

SYNCHRONIZER thread:

do forever:
    Wait for  M_REQ_SYN message for EX1.Params
    Reply with M_SYNCH message for EX1.Params


FLOODER thread:

do forever:
    Send M_FLOOD message for EX1.Params
    sleep() #sleep time depends on application scenario
```

```
GARBAGE_COLLECTOR thread:

do forever:
    Search resource_pool for adjacent resources
    Merge adjacent resources
    sleep() #sleep time depends on application scenario

<CODE ENDS>
```

Authors' Addresses

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

Email: brian.e.carpenter@gmail.com

Laurent Ciavaglia
Nokia
Villarceaux
91460 Nozay
France

Email: laurent.ciavaglia@nokia.com

Sheng Jiang
Huawei Technologies Co., Ltd
Q14 Huawei Campus
156 Beiqing Road
Hai-Dian District
Beijing
100095
China

Email: jiangsheng@huawei.com

Pierre Peloso
Nokia
Villarceaux
91460 Nozay
France

Email: pierre.peloso@nokia.com

Network Working Group B. E. Carpenter
Internet-Draft Univ. of Auckland
Intended status: Informational S. Jiang
Expires: 13 July 2020 B. Liu
Huawei Technologies Co., Ltd
10 January 2020

# Transferring Bulk Data over the GeneRic Autonomic Signaling Protocol (GRASP)

draft-carpenter-anima-grasp-bulk-05

## Abstract

This document describes how bulk data may be transferred between Autonomic Service Agents via the GeneRic Autonomic Signaling Protocol (GRASP). Although not an equivalent of a file transfer protocol, such a technique may be used for non-urgent transfer of data too large to fit into a normal GRASP message.

## Status of This Memo



## Copyright Notice

Table of Contents



## 1. Introduction

The document [I-D.liu-anima-grasp-distribution] discusses how information may be distributed within the secure Autonomic Networking Infrastructure (ANI) [I-D.ietf-anima-reference-model]. Specifically, it describes using the Synchronization and Flood Synchronization mechanisms of the GeneRic Autonomic Signaling Protocol (GRASP) [I-D.ietf-anima-grasp] for this purpose as well as proposing GRASP extensions to support a publish/subscribe model. However, those mechanisms are limited to distributing GRASP Objective Options contained in messages that cannot exceed the GRASP maximum message size of 2048 bytes. This places a limit on the size of data that can be transferred in a Synchronization or Flood operation.

There are scenarios in autonomic networks where this restriction is a problem. One example is the distribution of network policy in lengthy formats such as YANG or JSON. Another case might be an Autonomic Service Agent (ASA) uploading a log file to the Network Operations Center (NOC). A third case might be a supervisory system downloading a software upgrade to an autonomic node. A related case might be installing the code of a new or updated ASA to a target node (see the discussion of ASA life cycles in [I-D.carpenter-anima-asa-guidelines]).

Naturally, an existing solution such as a secure file transfer protocol or secure HTTP might be used for this. Other management protocols such as syslog [RFC5424] or NETCONF [RFC6241] might also be used for related purposes, or might be mapped directly over GRASP. The present document, however, applies to any scenario where it is preferable to re-use the autonomic networking infrastructure itself to transfer a significant amount of data, rather than install and configure an additional mechanism. The basic model is to use the GRASP Negotiation process to transfer and acknowledge multiple blocks of data in successive negotiation steps, thereby overcoming the GRASP message size limitation.

The emphasis is placed on simplicity rather than efficiency, high throughput, or advanced functionality. For example, if a transfer gets out of step or data packets are lost, the strategy is to abort the transfer and try again. In an enterprise network with low bit error rates, and with GRASP running over TCP, this is not considered a serious issue. Clearly, a more sophisticated approach could be designed but if the application requires that, existing protocols could be used, as indicated in the preceding paragraph.

This is an informational description of a class of solutions. Standards track solutions could be published as detailed specifications of the corresponding GRASP objectives.

## 2. General Method for Bulk Transfer

As for any GRASP operation, the two participants are considered to be Autonomic Service Agents (ASAs) and they communicate using a specific GRASP Objective Option, containing its own name, some flag bits, a loop count, and a value. In bulk transfer, we can model the ASA acting as the source of the transfer as a download server, and the destination as a download client. No changes or extensions are required to GRASP itself, but compared to a normal GRASP negotiation, the communication pattern is slightly asymmetric:

1. The client first discovers the server by the GRASP discovery mechanism (M_DISCOVERY and M_RESPONSE messages).

2. The client then sends a GRASP negotiation request (M_REQ_NEG message). The value of the objective expresses the requested item (e.g., a file name - see the next section for a detailed example).

3. The server replies with a negotiation step (M_NEGOTIATE message). The value of the objective is the first section of the requested item (e.g., the first block of the requested file as a raw byte string).

4. The client replies with a negotiation step (M_NEGOTIATE message). The value of the objective is a simple acknowledgement (e.g., the text string 'ACK').

The last two steps repeat until the transfer is complete. The server signals the end by transferring an empty byte string as the final value. In this case the client responds with a normal end to the negotiation (M_END message with an O_ACCEPT option).

Errors of any kind are handled with the normal GRASP mechanisms, in particular by an M_END message with an O_DECLINE option in either direction. In this case the GRASP session terminates. It is then the client's choice whether to retry the operation from the start, as a new GRASP session, or to abandon the transfer.

The block size must be chosen such that each step does not exceed the GRASP message size limit of 2048 bits.

This approach is safe since each block must be positively acknowledged, and data transfer errors will be detected by TCP. If a future variant of GRASP runs over UDP, the mandatory UDP checksum for IPv6 will detect such errors. The method does not specify retransmission for failed blocks, so the ASA that detects the error must signal the error as above.

An observant reader will notice that the GRASP loop count mechanism, intended to terminate endless negotiations, will cause a problem for large transfers. For this reason, both the client and server must artificially increment the loop count by 1 before each negotiation step, cancelling out the normal decrement at each step.

If network load is a concern, the data rate can be limited by inserting a delay before each negotiation step, with the GRASP timeout set accordingly. Either the server or the client, or both, could insert such a delay. Also, either side could use the GRASP Confirm Waiting (M_WAIT) message to slow the other side down.

The description above concerns bulk download from a server (responding ASA) to a client (requesting ASA). The data transfer could also be in the opposite (upload) direction with minor modifications to the procedure: the client would send the file name and the data blocks, and the server would send acknowledgements.

## 3. Example for File Transfer

This example describes a client ASA requesting a file download from a server ASA.

Firstly we define a GRASP objective informally:

```
["411:mvFile", 3, 6, value]
```

The formal CDDL definition [RFC8610] is:

```
mvfile-objective = ["411:mvFile", objective-flags, loop-count, value]

objective-flags = ; as in the GRASP specification
loop-count = ; as in the GRASP specification
value = any
```

The objective-flags field is set to indicate negotiation.

Dry run mode must not be used.

The loop-count is set to a suitable value to limit the scope of discovery. A suggested default value is 6.

The value takes the following forms:

* In the initial request from the client, a UTF-8 string containing the requested file name (with file path if appropriate).

* In negotiation steps from the server, a byte string containing at most 1024 bytes. However:

  - If the file does not exist, the first negotiation step will return an M_END, O_DECLINE response.

  - After sending the last block, the next and final negotiation step will send an empty byte string as the value.

* In negotiation steps from the client, the value is the UTF-8 string 'ACK'.

Note that the block size of 1024 is chosen to guarantee not only that each GRASP message is below the size limit, but also that only one TCP data packet will be needed, even on an IPv6 network with a minimum link MTU.

We now present outline pseudocode for the client and the server ASA. The API documented in [I-D.ietf-anima-grasp-api] is used in a simplified way, and error handling is not shown in detail.

Pseudo code for client ASA (request and receive a file):

```
requested_obj = objective('411:mvFile')
locator = discover(requested_obj)
requested_obj.value = 'etc/test.pdf'
received_obj = request_negotiate(requested_obj, locator)
if error_code == declined:
    #no such file
    exit

file = open(requested_obj.value)
file.write(received_obj.value) #write to file
eof = False
while not eof:
    received_obj.value = 'ACK'
    received_obj.loop_count = received_obj.loop_count + 1
    received_obj = negotiate_step(received_obj)
    if received_obj.value == null:
        end_negotiate(True)
        file.close()
        eof = True
    else:
        file.write(received_obj.value) #write to file

#file received
exit
```

Pseudo code for server ASA (await request and send a file):

```
supported_obj = objective('411:mvFile')
requested_obj = listen_negotiate(supported_obj)
file = open(requested_obj.value) #open the source file
if no such file:
    end_negotiate(False) #decline negotiation
    exit

eof = False
while not eof:
    chunk = file.read(1024) #next block of file
    requested_obj.value = chunk
    requested_obj.loop_count = requested_obj.loop_count + 1
    requested_obj = negotiate_step(requested_obj)
    if chunk == null:
        file.close()
        eof = True
        end_negotiate(True)
        exit
    if requested_obj.value != 'ACK':
        #unexpected reply...
```

## 4. Loss Detection

The above description and example assume that GRASP is implemented over a reliable transport layer such as TCP, such that lost or corrupted messages are not likely. Rarely, an error might be detected via a missing ACK, in which case the transfer would be aborted and restarted. In the event that GRASP is implemented over an unreliable transport layer such as UDP, it would be possible to add a block number to both the data block and acknowledgement objectives, so that missing blocks can be retransmitted, or duplicate blocks can be ignored. For example, the objective in Section 3 would become:

```
mvfile-objective = ["411:mvFile", objective-flags, loop-count, value]

objective-flags = ; as in the GRASP specification
loop-count = ; as in the GRASP specification
value = [block-number, any]
block-number = uint
```

It would also be necessary for the transport layer to detect data errors, for example by enabling UDP checksums.

## 5. Maximum Transmission Unit

In an IPv6 environment, a minimal MTU of 1280 bytes can be assumed, and assuming that high throughput is not a requirement, bulk transfers can be designed to match that MTU. However, there are environments where the underlying physical MTU is much smaller. For example, on an IEEE 802.15.4 network it may be less than 100 bytes [RFC4944]. Even in a 5G network, the Transport Block Size may be quite small, depending on the radio parameters. In such a case, a bulk transfer solution has several choices:

1. Accept the overhead of fragmentation in an adaptation layer, and therefore assume a network-layer MTU of 1280 bytes. Indeed, the presence of such an adaptation layer may be impossible to detect.

2. Attempt to determine the actual MTU available without lower-layer fragmentation. This however will be impossible without using low-level functions of the socket interface.

3. Attempt to determine a message size that provides optimum performance, by some sort of trial-and-error solution.

These complexities suggest that using a GRASP-based mechanism is unlikely to be optimal in environments with a very small physical MTU.

## 6. Pipelining

The above description and example descibe a simple handshake model where each block is acknowledged before the next block is sent. For the scenarios discussed in Section 1, this should be acceptable. Therefore we do not suggest adding a pipelining or windowing mechanism. If high throughput is required, a conventional file transfer protocol should be used.

## 7. Other Considerations

If multiple transfers are requested simultaneously, each one will proceed as a separate GRASP negotiation session. The ASA acting as the server must be coded accordingly, like any ASA that needs to handle simultaneous sessions [I-D.carpenter-anima-asa-guidelines].

Bulk transfer might become a utility function for use by various ASAs, such as those supporting YANG or JSON distribution, log file uploads, or code downloads. In this case some form of user space API for bulk transfer will be required. This could be in the form of an inter-process communication call between the ASA in question and the ASA implementing the bulk transfer mechanism. The details are out of scope for this document.

## 8. Possible Future Work

The simple file transfer mechanism described above is only an example. Other application scenarios should be developed.

The mechanism described in this document is suitable for simple unicast scenarios where GRASP runs over TCP and can be treated as a reliable protocol. A more sophisticated approach would be needed in at least two cases:

1. A scenario where GRASP runs over UDP, where error detection and retransmission would be essential.

2. A scenario where multicast data distribution is required, so that a mechanism such as Trickle [RFC6206] would be appropriate.

These solutions might also require extensions to the GRASP protocol itself.

## 9. Implementation Status [RFC Editor: please remove]

A prototype open source Python implementation of simple file transfer has been used to verify the mechanism described above. It may be found at https://github.com/becarpenter/graspy/blob/master/getter.py and https://github.com/becarpenter/graspy/blob/master/pusher.py .

## 10. Security Considerations

All GRASP transactions are secured by the mandatory security substrate required by [I-D.ietf-anima-grasp]. No additional security issues are created by the application of GRASP described in this document.

## 11. IANA Considerations

This document makes no request of the IANA.

## 12. Acknowledgements

Thanks to Joel Halpern and other members of the ANIMA WG.

## 13. References

### 13.1. Normative References

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", Work in Progress, Internet-Draft, draft-ietf-anima-grasp-15, 13 July 2017, <https://tools.ietf.org/html/draft-ietf-anima-grasp-15>.

[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>.

### 13.2. Informative References

[I-D.carpenter-anima-asa-guidelines]
Carpenter, B., Ciavaglia, L., Jiang, S., and P. Pierre, "Guidelines for Autonomic Service Agents", Work in Progress, Internet-Draft, draft-carpenter-anima-asa-guidelines-07, 6 July 2019, <https://tools.ietf.org/html/draft-carpenter-anima-asa-guidelines-07>.

[I-D.ietf-anima-grasp-api]
Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)", Work in Progress, Internet-Draft, draft-ietf-anima-grasp-api-04, 6 October 2019, <https://tools.ietf.org/html/draft-ietf-anima-grasp-api-04>.

[I-D.ietf-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", Work in Progress, Internet-Draft, draft-ietf-anima-reference-model-10, 22 November 2018, <https://tools.ietf.org/html/draft-ietf-anima-reference-model-10>.

[I-D.liu-anima-grasp-distribution]
Liu, B., Xiao, X., Hecker, A., Jiang, S., and Z. Despotovic, "Information Distribution in Autonomic Networking", Work in Progress, Internet-Draft, draft-liu-anima-grasp-distribution-13, 12 December 2019, <https://tools.ietf.org/html/draft-liu-anima-grasp-distribution-13>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, DOI 10.17487/RFC5424, March 2009, <https://www.rfc-editor.org/info/rfc5424>.

[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>.

## Appendix A. Change log [RFC Editor: Please remove]

draft-carpenter-anima-grasp-bulk-05, 2020-01-10:

* Minor technical clarifications.
* Converted to v3 format.

draft-carpenter-anima-grasp-bulk-04, 2019-07-03:

* Updated description of very small link-layer MTU issue.
* Clarified informational status, updated reference.

draft-carpenter-anima-grasp-bulk-03, 2019-01-07:

* Added future work section, implementation status.

draft-carpenter-anima-grasp-bulk-02, 2018-06-30:

* Update reference, fix TBDs.

draft-carpenter-anima-grasp-bulk-01, 2018-03-03:

* Updates after IETF100 discussion.

draft-carpenter-anima-grasp-bulk-00, 2017-09-12:

* Initial version.

Authors' Addresses

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

Email: brian.e.carpenter@gmail.com

Sheng Jiang
Huawei Technologies Co., Ltd
Q14 Huawei Campus
156 Beiqing Road
Hai-Dian District
Beijing
100095
China

Email: jiangsheng@huawei.com

Bing Liu
Huawei Technologies Co., Ltd
Q14 Huawei Campus

156 Beiqing Road
Hai-Dian District
Beijing
100095
China

Email: leo.liubing@huawei.com

ANIMA WG T. Eckert
Internet-Draft Huawei
Intended status: Standards Track Jul 1, 2018
Expires: January 2, 2019

# DNS-SD compatible service discovery in GRASP
draft-eckert-anima-grasp-dnssd-01

## Abstract

DNS Service Discovery (DNS-SD) defines the common framework for applications to announce and discover services. This includes service names, service instance names, common parameters for selecting a service instance (weight, priority) as well as service specific parameters.

GRASP is intended to also be used for service discovery. Reinventing service discovery for GRASP with a similar set of fetures would result in duplication of work. Therefore, this document defines how to use GRASP to announce and discover services in a way that inherits DNS-SD features and also tries to be compatible in spirit as much as possibel while still maintaining the intended simplicity of GRASP.

The goal of this document is to permit defining service and their parameters once and then use that in GRASP, mDNS and (unicast) DNS. Future work can also define DNS-SD <-> GRASP gateway functions.

In support of service discovery, this document also defines name discovery and schemes for reuseable elements in GRASP objectives which are designed to be extensible so that future work that identifies elements required across multiple objectives do not need to define a scheme how to do this.

## Status of This Memo

This Internet-Draft will expire on January 2, 2019.

Copyright Notice



Table of Contents



## 1. Overview

DNS Service Discovery (DNS-SD) defines the common framework for applications to announce and discover services. This includes service names, service instance names, common parameters for selecting a service instance (weight, priority) as well as service specific parameters.

GRASP is intended to also be used for service discovery. Reinventing service discovery for GRASP with a similar set of fetures would result in duplication of work. Therefore, this document defines how to use GRASP to announce and discover services in a way that inherits DNS-SD features and also tries to be compatible in spirit as much as possibel while still maintaining the intended simplicity of GRASP.

The goal of this document is to permit defining service and their parameters once and then use that in GRASP, mDNS and (unicast) DNS. Future work can also define DNS-SD <-> GRASP gateway functions.

GRASP exists as so-called GRASP-Domains, which are networks across which GRASP is run. This document primarily defines how to perform service discovery across such a domain leveraging GRASPs options to perform unsolicited flooding of announcements or flooding of requests and finding the closest service instances. The initial use case of this document is to support what in DNS-SD is done via mDNS but in larger networks - GRASP-Domains. Beside the efficient flooding, GRASP provides reliability and security (depending on the so called substrate used by GRASP, such as the autonomic control plane - ACP). Providing compatibility with existing mDNS service announcer or clients is possible, but not described in this version of the document.

The encoding of information choosen in this document does not try to use GRASP solely as a transport layer, but to also leverage the CBOR structure of GRASP messages to natively encode the message elements required for services in a way that is most simple - instead of using GRASP only as e.g.: an encapsulation of otherwise unchanged DNS message encodings. This is done to minimize the amount of coding required (and not require any DNS code unless future gateway functions are requireed), to increase the simplicity, minimize the amount of data on the wire and allow easier extensibility. On the downside, the mechanisms provided here do not cover the whole slew of possible options of DNS/DNS-SD, but instead only those deemed to be required. Others can be added later.

In support of service discovery, this document also defines name discovery and schemes for reuseable elements in GRASP objectives which are designed to be extensible so that future work that identifies elements required across multiple objectives do not need to define a scheme how to do this.

## 2. Specification (Normative)

### 2.1. Service and Name Objectives

Unsolicited, flooded announcements (M_FLOOD) in GRASP and solicited flooded discovery (M_DISCOVERY) operate on the unit of GRASP objective-names. Therefore a scheme is required to indicate services via objective-names. Note: future work may wants to reuse the encodings related to services (defined below in this document) inside other (multicast or unicast only) objective exchanges, in which case the service names are not impacted.

When an objective is meant to be solely about a service name as defined and registered according to RFC6335, the objective MUST uses an objective-name of SRV.<service-name>. This naming scheme allows to avoid creating duplicate and potentially inconsistent registration of names for those objectives vs. registrations done for example for DNS-SD. The primary use case for this naming scheme are therefore service names that are intended to be used in both DNS-SD and GRASP.

When an objective is meant announcement and discovery of a DNS compatible <name> such as "www-internal" in "www-internal.example.com", the objective SHOULD use an objective-name of NAME.<name>. See Section 2.3.3for more details.

See Section 5 for the detailled IANA asks relating to these definitions.

### 2.2. Objective Value Reuseable Elements Structure

Because service discovery, as explained in the prior section, needs to utilize different objectives, it requires cross-objective standardized encoding of the elements of services. GRASP did not define standardized message elements for the message body (called "objective-value") of GRASP messages. Therefore, this document introduces such a feature.

[RFC-editor: please remove all occurances of XXXX in rfcXXXX with the RFC number assigned to this document and remove this edit note.]

```
objective-value  /= { 1*elements }
elements        //= ( @rfcXXXX: { 1*relement } )

relement  = ( relement-codepoint => relement-value )
relement-codepoint = uint
relement-value     = any
```

If an objective wants to use reuseable elements, the objective-value MUST be a CBOR map and the reuseable elements are found under the key "@rfcXXXX". Objectives that do not want reuseable elements as defined here can use any objective-value format including a CBOR map, but they can not use the "@rfcXXXX" key if they use a map. This approach was choosen as the hopefully least intrusive mechanism given how by nature all of "objective-value" is meant to be defined by individual objective definitions.

The value of "@rfcXXXX" is a map of reuseable elements. Each relement has an IANA registered element-name and codepoint (see Section 5). The element-name is for documentation purposes only, CBOR encodings only use the numeric codepoint for encoding efficiency to minimize the risk for this solution to not be applicable to low-bitrate neworks such as in IoT.

Format and semantic of the relement-value is determined by the specification of the reuseable element as is the fact whether more than one instances of the same reuseable element are permitted.

Reuseable elements SHOULD be defined to be extensible. The methods used depend on the complexity of the element and the likely need to extend/modify the element with backward or non-backward compatible information. The following is a set of initial options to choose from:

Element values that are a map MUST permit and reserve key value 0 (numerical) for private extensions of the element defined by the individual objective.

Element values that are a map MUST NOT use bareword key values starting with a "_". These too are for private extensions defined by the individual objective.

Element values SHOULD be defined so that additional keys in maps and additional elements at the end of arrays can be ignored by prior versions of the definition. Whenever a newer definition is made for an element where this rule is violated, the element SHOULD be changed in a way for older version recipients to recognize that it is not compatible with it.

One method to indicate compatibility is a traditional version "<mayor>.<minor>". Within the same <mayor> version number, increasing <minor> version numbers must be backward compatible. Different <mayor> version numbers are not expected to be compatible with each other. If they are, then this can be indicated by including multiple version numbers.

A compressed form of version compatibility information is the use of a simple bitmask element where each bit indicates a version that the represented data is compatible with.

### 2.3. Reuseable Elements

#### 2.3.1. Sender Loop Count

```
relement-codepoint //= ( &(sender-loop-count:1) => 1..255 )
```

Sender-loop-count is set by the sender of an objective message to the same value as the loop-count of the message. On receipt, distance = ( sender-loop-count - loop-count ) is the distance of the sender from the receiver in hops. This element can be used for informational purposes in M_FLOOD and M_DISCOVERY messages and may be required to be used in these messages by the specification of other elements (such as the service element described below). This element MUST occur at most once. If a receiver expects to use the distance but sender-loop-count was not announced, then distance SHOULD be assumed to be 255 by the receiver.

#### 2.3.2. Service Element

The srv-element (service element) is a reuseable element to request or announce a service instance or to request and list service instance names.

```
relement-codepoint //= ( &(srv-element:2) => context-element )

context-element  =  {
     ?( &(private:0)      => any),
     ?( &(msg-type:1      => msg-type),
     ?( &(service:2)      => tstr),
     *( &(instance:3)     => tstr),
     ?( &(domain:4)       => tstr),
     ?( &(priority:5)     => 0..65535 ),
     ?( &(weight:6)       => 0..65535 ),
     *( &(kvpairs:7)      => { *(tstr: any) },
     ?( &(range:8)        => 0..255 ),
     *( &(clocator:9)     => clocator),
    }
clocator = [ context, locator-option ]
context = cstr
locator-option = ; from GRASP

msg-type = &( describe: 0, describe-request:1,
             enumerate:2, enumerate-request:3 )
```

Service: A service name registered according to RFC6335. If it is not present, then objective-name MUST be SRV.<service-name> where <service-name> is the service-name.

Instance: The <Instance> of a DNS-SD Service Instance Name ( <Instance> . <Service> . <Domain>). It is optional, see Section 3.2.

Domain: The equivalent of the <Domain> field of a DNS-SD Service Instance Name. If domain is not present, this is equivalent to ".local" in DNS (as introduced by mDNS) and implies the unnamed "local" domain, which is the GRASP domain across which the message is transmitted.

Priority, Weight: Service Instance selection criteria as defined in RFC2782. If either one is not present, its value defaults to 0.

Kvpairs: Map of key/value pairs that are service parameters in the same format as the key/value pairs in TXT field(s) of DNS-SD TXT records as defined in RFC6763, section 6.3.

Range: Allows to flexibly combine distance and priority/weight based service selection according to the definition of distance in Section 2.3.1.

If min-distance is the distance of the closest service announcer, and min-range the range announced by it, then the recipient MUST consider the priority/weight of all service announcers that are not further away than (min-distance + min-range). If not included, range defaults to 255.

If range is announced, the sender-loop-count element MUST also be announced.

Clocator: The "contextual locator" allows to indicate zero or more locators for the indicated service instance. The context element indicates in which context the locator-option is to be resolved. The reserved context value of "" (empty string) indicates the GRASP domain used, aka: the "local" context in which the service announcement is made. The reserved context value of "0" indicates the default routing context of the announcing node. This is often called "global table", "VRF 0" or "default VRF" on nodes using the "VRF" abtraction. Any other value is a string specifying a context such as another VRF.

The mechanism by which originator and recipient of the srv-element agree on common naming for contexts is outside the scope of this specification. The context therefore allows to indicate locators both for the context through which the GRASP message distributed the srv-element (GRASP domain) as well as that for other contexts. Assume the GRASP domain is the ACP, then clocators in ACP would have a context of "", clocators in the global routing table (part of the data-plane) a context of "0", and clocators on other VRFs (also part of data-plane) a clocator that is their sring name.

If no locators are indicated, then the locator of the service(s) is the optional locator-option of the GRASP message in which the objective is contained meant to be used for the service(s) indicated and the clocator implied is "".

If locator(s) are indicated, the messages location-option must be ignored for the service (but may be necessary to be present for other purposes of the objective).

Msg-type Type (aka: intention) of the srv-element. If not present, it is assumed to be "describe".

Describe: Describes one service instance. At least one clocator is required for a positive response, all other fields are permitted, but optional. "Describe" is used in M_FLOOD for unsolicited announcements of services (flooded), in M_RESPONSE messages for solicited announcements of a service and in M_NEGOTIATE for negotiated announcements (both unicasted). If clocator is not included, then all fields except service and instance (and msg-type and private) must not be included and the srv-element provides a negative reply: No information about this service/ service instance. This is only permitted in unicasted "describe" messages.

Describe-request: Request for a "describe" reply. It is used in M_DISCOVERY (flooded) for solicited discovery of services or in M_REQ_SYN (unicasted) for negotiated discovery of service instance(s). In "describe-request", only service is mandatory (but can be provided via the objective-name field of the message), and domain is optional. "Instance" is optional. If provided, then the recipient is asked to provide information about the named instance only. All other fields of srv-element are to be ignored by the receiver in this specification, but a semantic for setting them may be introduced in followup work, specifically to filter replies by the indiciated fields.

"Describe-request" without instance MAY be answered by "Enumerate" (see below) if the responder has so many instances that it thinks the initiator should rather first select one or fewer instances and ask for their description. The sender of te "Describe-request" MUST be prepared to accept that answer and as necessary follow up with "Describe-request" with the instance names of interest.

Enumerate: Used in the same GRASP messages as "describe", but instead of providing information about one service instance, it is listing service instance names. The purpose of enumerate is the same as browsing a service in DNS-SD. It would be followed by some human or automated selection of one or more instances and then a "describe" M_REQ_SYN request for those instances sent to the source of the "enumerate" to learn about the locators and other parameters of the service instances.

In this specification, all fields other than service, instance and domain (and msg-type and private) must be unset in "enumerate".

Enumerate-request: Requests an "enumerate" reply. It is used in the same way as "Describe-request" except that instance would usually not be set (because in that case it is more useful to send a "Describe-request").

### 2.3.3. Name Element

The NAME,<name> elements is meant to provide basic name resolution comparable to mDNS name resolution for GRASP domains where this is desirable and no better name resolution exist - for example in the ACP where there is no requirement for DNS.

Because the GRASP service lookup (unlike) DNS does not mandate that nodes have names (not even service instance names), the use of names is primarily meant to support legacy software. New designs should instead look up only services and service instance names, and nodes should announce their names as service instance names for the services they offer:

For example consider a GRASP (ACP) domain of "example.com". The node providing some "www" service could have a name "www-internal" which means GRASP objective NAME.www-internal, that objective value would include primarily the nodes IP address(es) and the port number for the www service would have to be guessed (80). Better, the node would announce GRASP objective SRV.www and the objective value would include the service instance name www-internal and the (TCP) port information (80 or a non-default port).

```
relement-codepoint //= ( &(name-element:3) => context-element )

context-element //= {
     *( &name:10)         => tstr),
    }

ipv6-address-option = [O_IPv4_ADDRESS, ipv6-address]
ipv4-address-option = [O_IPv6_ADDRESS, ipv6-address]
locator-option /= ipv4-address-option
locator-option /= ipv6-address-option
```

Name information is carried in the name-element relement. It is a context-element like the one used for srv-element except that it adds the name component and that it does not permit the service and instance components and that it allows only describe and describe-request values in the msg-type. Clocators MUST use the ipv6-address-option or ipv4-address-option in the locator-option component.

TBD: Unclear if/how we should best formalize the differences in the context element permitted information between services and names. The above is quite informal.

Priority, weight, kvpairs, range (and of course private) MAY be used in describe messages to support multiple instances of the same name, as used for name anycast/prioritycast.

Nodes may have multiple names. These can be listed in the name component. If a nodes names have the notion of a primary name and secondary names then the primary name should be the first in the list of names. In DNS-SD, the name pointed to by CNAME RRs can be considered to be the primary name. A describe-request for a non-primary name SHOULD return in the list of names the requested name and the primary name.

Note that there is no reverse lookup defined in this version of the document (no lookup from IP address to name).

# 3. Explanations (Informative)

## 3.1. Using GRASP service announcements

TBD: This section contains a range of details that should become normative in later versions.

This section provides a step by step walk-through of how to use GRASP service announcements and compares it to DNS-SD.

The most simple method to use GRASP service discovery is to select (and if still neessary, register) a <service-name> and start one or more agents (e.g.: ASAs) announcing their service instance(s) via GRASP. At minimum, an agent should periodically (default 60 seconds) announce the service instance via GRASP M_FLOOD messages as an objective SRV.<service-name> with a srv-element and a sender-loop-count element (default 255). The ttl of the GRASP message should be 3.5 times the announcement period, e.g.: 210000 msec.

Consumers of the service will use GRASP to learn of the service instances and select one. This approach is most similar to the use of DNS-SD with mDNS except that the scope of the announcement is a whole GRASP domain (such as the ACP) as opposed to a single IP subnet in mDNS and that mDNS primarily relies on request & reply but in its standard not on periodic unsolicited announcements. We describe here the unsolicited flooding option via M_FLOOD first because it is recommended for services with a dense population of service consumers and it is most simple to describe.

On the service announcer, the parameters priority, weight and range of the service instance can be selected from intent or configuration - or left at default. The default range 255 will result in selection of a random target of the service like in DNS-SD. Setting priority/

weight allows to prioritize and weigh the selection as in DNS-SD. Setting range to 0 allows to select the closest target, priority/ weight are only compared between targets of the same shortest distance. Distance based options are not available in DNS-SD because it does not expect that network distance is available to arbitrary DNS-SD client. It is available to GRASP clients though. Using 0 < range < 255 allows for a hybrid priority/weight and distance based service selection (e.g.: Select the highest priority instance within a range of 5 hops).

If the service is a non-GRASP service, then the result of the service discovery has to be a transport locator to which the client can open a connection and talk the protocol implied by the service. This transport locator(s) have to be put into the clocator parameter. The context of the clocator would normally be "", aka: the transport locator is in the IP reachability associated with the GRASP domain (e.g.: IPv6 of the ACP for ACP GRASP domain).

If an ACP service is announced via ACP GRASP, then the locator(s) can be O_IPv6_LOCATOR or O_FQDN_LOCATOR. The O_IPv6_LOCATOR is used if the service is defined to be available via some transport layer port (TCP, UDP or other). The determination of the actual transport connection to be used is the same as in DNS-SD: If the transport protocol is not TCP or UDP, it has to be implied by the specification of <service-name> or can be detailled in kvpairs which carries the same information as DNS-TXT TXT RRs of the service. Alternatively, the transport-proto field of the locator can contain any valid IP protocol directly (TBD), which is not possible in DNS-SD.

Like DNS-SD, service discovery via GRASP does not require allocation and use of well-known ports for services. Unlike DNS-SD, there is no need in GRASP to define service instance names or target names. In DNS SD, PTR RRs resolve from a service name to a set of service instance named. SRV and TXT RRs resolve from service instance names to service instance parameters including the target. A target is the DNS host name of the service instance. It gets resolved via A/AAAA RRs to IPv4/IPv6 addresses of the targ. In GRASP service discovery, host names are not used. Service instance names are optional too. Service instance names are useful for human diagnostics and human selection of service instances. In fully automated environments, they can be are less important. For diagnostic purposes, it is recommended to give service instances service instance names in GRASP service announcements.

A locator with O_URI_LOCATOR type can be used in GRASP to indicate a URI for the transport method for a service instance. If the URI includes a host part, care must be taken to use only IP addresses in the host part if the context of the GRASP domain does not support host name resolution - such as the ACP - or to use the GRASP name resolution mechanisms described elsewhere in this document. And that the addresses indicated are also reachable in the GRASP domain. For example, in service announcemenets across a DULL GRASP domain, only the IPv6 link-local addresses on that subnet must be used (this applies equally when using the O_IPv6_LOCATOR).

Instead of using M_FLOOD to periodically announce service instances, M_DISCOVERY can be used to actively query for service instances. The msg-type type must then be "describe-request". Because no periodic flooding is necessary, this solution is more lightweight for the network when the number of requesting clients is small. Note though that the M_DISCOVERY will terminate as soon as a provider of the objective is found, so the service instances found will be based on distance and therefore selection of instance by priority and weight will not work equally well as with M_FLOOD. Consider for example a central service instance in the NOC that should always be used (for example for centralized operational diagnostics) unless the WAN connection is broken, in which case distributed backup service instances should be used. With the current logic of M_DISCOVERY this is not possible.

## 3.2. Further comparison with DNS-SD

Neither the GRASP SRV.* objective-name, the service name nor any other parameter explicitly indicate the second label "_tcp" or "_udp" of DNS-SD entries. DNS-SD, RFC6763 explains how this is an unnecessary, historic artefact.

This version of the document does not define an equivalent to "_sub" structuring of service enumeration.

This version of the document does not define mechanisms for reverse resolution of arbitrary services: An inquirer may unicast M_SYNC_REC to a node with a series of objectives with specific service names of interest and describe-request, but there is no indication of "ANY" service.

## 3.3. Open Issues

TBD: Examine limitations mentioned in "in this version of the text/ document".

TBD: The GRASP specification does currently only permit TCP and UDP for the transport-proto element. This draft should expand the GRASP definitions to permit any valid IP protocol. We just need to decide whether this should only apply to the locator in the srv element or also retroative to the locator-option in GRASP messages (maybe not there ?).

TBD: A fitting CBOR representation for a kvpair key without value needs to be specified so that it can be distinguished from an empty value as outlined in RFC6763 section 6.4.

TBD: In this version, every service/service-instance is an element by itself. Future versions of this document may add more encoding options to allow more compact encoding of recurring fields.

TBD: Is there a way in CDDL to formally define the string names of the relement-codepoint’s ?

## 4. Security Considerations

TBD.

## 5. IANA Considerations

This document requests a new "GRASP Objective Value Standard Elements" table in the GRASP Parameter Registrar. The values in this table are names and a unique numerical value assigned to each name. Future values MUST be assigned using the RFC Required policy defined by [RFC8126]. The numerical value is simply to be assigned sequentially. The following initial values are assigned by this document:

sender-loop-count 1 [defined in rfcXXXX]

srv-element 2 [defined in rfcXXXX]

name-element 3 [defined in rfcXXXX]

This document updates the handling of the "GRASP Objective Names" Table introduced in the GRASP IANA considerations as follows:

Assignments for objective-names of the form "SRV.<text>" and "NAME.<text>" are special.

Assignment of "SRV.<text>" can only be requested if <text> is also a registered service-name according to RFC6335. The specification required for registration of a "GRASP Objective Name" MUST declare that the intended use of the objective name in GRASP is intended to be compatible with the indended use of the registered service name.

Registration of "SRV.<text>" in the "GRASP Objective Name" table is optional, but recommended for all new service-names that are meant to be used with GRASP. Non-registration can for example happen with DNS-SD <-> GRASP gateways that inject pre-existing service-names into GRASP. Note that according to the GRASP RFC, registration is mandatory, so this exemption for "SRV.<text>" is also an update to that specification.

There MUST NOT be any assignment for objective names of the form "NAME.<text>". These names are simply used by GRASP nodes without registration (just like names in mDNS).

# 6. Acknowledgements

# 7. Change log [RFC Editor: Please remove]

## 7.1. 01 -

Only refreshing, no changes since -00.

## 7.2. 00 - Initial version

# 8. References

## 8.1. Normative References

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", draft-ietf-anima-grasp-15 (work in progress), July 2017.

[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>.

## 8.2. Informative References

[I-D.ietf-anima-autonomic-control-plane]
Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-ietf-anima-autonomic-control-plane-16 (work in progress), June 2018.

# Author’s Address

Toerless Eckert
Futurewei Technologies Inc.
2330 Central Expy
Santa Clara  95050
USA

Email: tte+ietf@cs.fau.de

No Working Group                                                A. Galis
Internet-Draft                                  University College London
Intended Status: Standards Track                             K. Makhijani
Expires: March 30, 2019                                             D. Yu
                                                                   B. Liu
                                                      Huawei Technologies
                                                       September 26, 2018

# Autonomic Slice Networking
## draft-galis-anima-autonomic-slice-networking-05

Abstract

This document describes the technical requirements and the related reference model for the intercommunication and coordination among devices in Autonomic Slicing Networking.  The goal is to define how the various elements in a network slicing context work and orchestrate together, to describe their interfaces and relations. While the document is written as generally as possible, the initial solutions are limited to the chartered scope of the WG.

Table of Contents

# 1 Introduction

The document "Autonomic Networking - Definitions and Design Goals" [RFC7575] explains the fundamental concepts behind Autonomic Networking, and defines the relevant terms in this space, as well as a high level reference model. This document defines this reference model with more detail, to allow for functional and protocol specifications to be developed in an architecturally consistent, non-overlapping manner. While the document is written as generally as possible, the initial solutions are limited to the chartered scope of the WG.

Most networks will run with some autonomic functions for the full networks or for a group of nodes [RFC7576] or for a group of slice networks while the rest of the network is traditionally managed.

The goal of this document is to focus on the autonomic slicing networking. [RFC7575] is focusing on fully or partially autonomic nodes or networks.

The proposed revised ANIMA reference model allows for this hybrid approach across all such capabilities. It enhances [ASN].

This is a living document and will evolve with the technical solutions developed in the ANIMA WG. Sections marked with (*) do not represent current charter items.

While this document must give a long term architectural view, not all functions will be standardized at the same time.

# 2. The Network Slicing Overall View

## 2.1. Key Terms and Context

A number of slice definitions were used in the last 10 years in distributed and federated testbed research [GENI], future internet research [ChinaCom09] and more recently in the context of 5G research [NGMN], [ONF], [IMT2020], [NGS-3GPP], [NS-ETSI]. Such definitions converge towards NS as group of components: Service Instance, Network Slice Instance, Resources and Slice Element Manager

In this draft we are using the following terms:

Logical resource - An independently manageable partition of a physical resource, which inherits the same characteristics as the physical resource and whose capability is bound to the capability of the physical resource. It is dedicated to a Network Function or shared between a set of Network Functions.

Virtual resource - An abstraction of a physical or logical resource, which may have different characteristics from that resource, and whose capability may not be bound to the capability of that resource

Network Function (NF) - A processing function in a network. It includes but is not limited to network nodes functionality, e.g. session management, mobility management, switching, routing functions, which has defined functional behaviour and interfaces. Network functions can be implemented as a network node on a dedicated hardware or as a virtualized software functions. Data, Control, Management, Orchestration planes functions are Network Functions.

Virtual Network Function (VNF) - A network function whose functional software is decoupled from hardware. One or more virtual machines running different software and processes on top of industry-standard high-volume servers, switches and storage, or cloud computing infrastructure, and capable of implementing network functions traditionally implemented via custom hardware appliances and middle. boxes (e.g. router, NAT, firewall, load balancer, etc.) Network Slicing (NS) refers to a managed group of subsets of resources, network functions / network virtual functions at the data, control, management/orchestration planes and services at a given time. Network slice is programmable and has the ability to expose its capabilities. The behaviour of the network slice realized via network slice instance(s). Network resources include connectivity, compute, and storage resources.

Network Slicing is end-to-end concept covering the radio and non-radio networks inclusive of access, core and edge / enterprise networks. It enables the concurrent deployment of multiple logical, self-contained and independent shared or partitioned networks on a common infrastructure platform

Network slicing represents logically or physically isolated groups of network resources and network function/virtual network functions configurations separating its behavior from the underlying physical network.

Network Slice Instance - An activated network slice. It is created based on network template. A set of managed run-time network functions, and resources to run these network functions, forming a complete instantiated logical network to meet certain network characteristics required by the service instance(s). It provides the network characteristics that are required by a service instance. A network slice instance may also be shared across multiple service instances provided by the network operator.

From a business point of view, a slice includes combination of all relevant network resources / functions / assets required to fulfill a specific business case or service, including OSS, BSS and DevOps processes.

From the network infrastructure point of view, slicing instances require the partitioning and assignment of a set of resources that can be used in an isolated, disjunctive or non- disjunctive manner.

Examples of physical or virtual resources to be shared or partitioned would include: bandwidth on a network link, forwarding tables in a network element (switch, router), processing capacity of servers, processing capacity of network or network clouds elements [SLICING]. As such slice instances would contain:

(i) a combination/group of the above resources which can act as a network,
(ii) appropriate resource abstractions,
(iii) capability exposure of abstract resources towards service and management clients that are needed for the operation of slices

The capability exposure creates an abstraction of physical network devices that would provide information and information models allowing operators to manipulate the network resources. By utilizing open programmable network interfaces, it would enable access to control layer by customer interfaces and applications.

The establishment of slices is both business-driven (i.e. slices are in support for different types and service characteristics and business cases) and technology-driven as slice is a grouping of physical or virtual) resources (network, compute, storage) which can act as a sub network and/or a cloud. A slice can accommodate service components and network functions (physical or virtual) in all network segments: access, core and edge / enterprise networks.

A complete slice is composed of not only various network functions which are based on virtual machines at C-RAN and C-Core, but also transport network resources that can be assigned to the slice at radio access/transport network. Different future businesses require different throughput, delay and mobility, and some businesses need very high throughput or/and low delay.

## 2.2. High Level Requirements

Slice creation: management plane create virtual or physical network functions and connects them as appropriate and instantiate them in the slice, which is a subnetworks.

The instance of slice management then takes over the management and operations of all the (virtualised) network functions and network programmability functions assigned to the slice, and (re-)configure them as appropriate to provide the end-to-end service.

A complete slice is composed of not only various network functions which are based on virtual machines at C-RAN and C-Core, but also transport network resources that can be assigned to the slice at radio access/transport network. Different future businesses [5GNS], [PER-NS] require different throughput, delay and mobility, and some businesses need very high throughput or/and low delay. Transport network shall provide QoS isolation, flexible network operation and management, and improve network utilization among different business.

(1) Separation from partition of the physical network: Network slicing represents logically or physically isolated groups of network resources and network function/virtual network functions configurations separating its behavior from the underlying physical network.

(2) QoS Isolation: Although traditional VPN technology can provide physical network resource isolation across multiple network segments, it is deemed far less capable of supporting QoS hard isolation, Which means QoS isolation on forwarding plane requires better coordination with management plane.

(3) Independent Management Plane: Like above, network isolation is not sufficient, a flexible and more importantly a management plane per instance is required to operate on a slice independently and autonomously within the constraints of resources allocated to the slice.

(4) Another flexibility requirement is that an operator can deploy their new business application or a service in network slice with low cost and high speed, and ensure that it does not affect existing of business applications adversely.

(5) Stringent Resource Characteristics: A Network Slicing aware infrastructure allows operators to use part of the network resources to meet stringent resource characteristics.

(6) Type of resources: Network Slice instance is a dedicated network

that is build and activated on an infrastructure mainly composed of, but not limited to, connectivity, storage and computing.

(7) Programmability: Operator not only can slice a common physical infrastructure into different logical networks to meet all kinds of new business requirements, but also can use SDN based technology to improve the overall network utilization. By providing a flexible programmable interface; the 3rd party can develop and deploy new network business rapidly. Further, if a network slicing can run with its own slice controller, this network slicing will get more granular control capability [I-D.ietf-anima-autonomic-control-plane] to retrieve slice status, and issuing slicing flow table, statistics fetch etc.

(8) Life cycle self-management: It includes creation, operations, re- configuration, composition, decomposition, deletion of slices. It would be performed automatically, without human intervention and based on a governance configurable model of the operators. As such protocols for slice set-up /operations /(de)composition / deletion must also work completely automatically. Self-management (i.e. self- configuration, self-composition, self-monitoring, self-optimisation, self-elasticity) is carried as part of the slice protocol characterization.

(9) Network slice Self-management: Network slices will need to be self-managed by automated, autonomic and autonomous systems in order to cope with dynamic requirements, such as flexible scalability, extensibility, elasticity, residency and reliability of an infrastructure. Network slices will need to be self-managed by automated, autonomic and autonomous systems in order to cope with dynamic requirements, such as scalability or extensibility of an infrastructure. A common information model describing uniformly the NS in a single and/or multiple domain would support such self-managed.

(10) Extensibility: Since the Autonomic Slice Networking Infrastructure is a relatively new concept, it is likely that changes in the way of operation will happen over time. As such new networking functions will be introduced later, which allow changes to the way the slices operate.

(11) Network Slice elasticity: A Network Slice instance has the mechanisms and triggers for the growth/shrinkage of all resources, and/or network and service functions as enabled by a common information model that explicitly provides for elasticity policies for scaling up/down resources.

(12) Multiple domains activation: Network slice instances are concurrently activated as multiple logical, self-contained and independent, partitioned network functions and resources on a specific infrastructure domain.

(13) Resource Exposure: Each network slice has the ability to dynamically expose and possibly negotiate the parameters that characterize an NS as enabled by a common information model that explicitly provides monitoring policies for all model descriptors.

(14) Network Tenants: Network slicing support tenants that are strongly independent on infrastructure as enabled by a common information model that explicitly provides for a level of tenants management for the resources dedicated to an instance of network slice.

(15) End-to-end Orchestration of Network Slicing: Coordinating underlay network infrastructure and service function resources. In the process of orchestration of network slice, resource registration and templates for network slice repository are needed.

## 3. Autonomic Slice Networking

This section describes the various elements in a network with autonomic functions, and how these entities work together, on a high level. Subsequent sections explain the detailed inside view for each of the autonomic network elements, as well as the network functions (or interfaces) between those elements.

From a business point of view, a slice includes a combination of all the relevant network resources, functions, and assets required to fulfill a specific business case or service, including OSS, BSS and DevOps processes.

From the network infrastructure point of view, network slice requires the partitioning and assignment of a set of resources that can be used in an isolated, disjunctive or non- disjunctive manner for that slice.

From the tenant point of view, network slice provides different capabilities, specifically in terms of their management and control capabilities, and how much of them the network service provider hands over to the slice tenant. As such there are two kinds of slices: (A) Inner slices, understood as the partitions used for internal services of the provider, retaining full control and management of them. (B)

Outer slices, being those partitions hosting customer services, appearing to the customer as dedicated networks.

Network Slicing lifecycle includes the management plane selecting a group of network resources (whereby network resources can be physical, virtual or a combination thereof); it connects with the physical and virtual network and service functions as appropriate, and it instantiates all of the network and service functions assigned to the slice. For slice operations, the control plane takes over governing of all the network resources, network and service functions assigned to the slice. It (re-) configures them as appropriate and as per elasticity needs, in order to provide an end-to-end service.

One expected autonomic Slice Networking function is the capability and resource Usability for a slice. Applications or services requiring information of available slice capabilities and resources are satisfied by abstracted resource view and control. Usability of capabilities and resources can be enabled either by resource publishing or by discovery. In the latter case, the service performs resource collection directly from the provider of the slice by using discovery mechanisms to get total information about the available resources to be consumed. In the former, the network provider exposes available resources to services (e.g., through a resource catalog) reducing the amount of detail of the underlying network.

Slice Element Manager (SEM) is installed for each control domain. Control domain is defined according to geographic location and control functions. Each SEM converts requirements from orchestrator into virtual resources and manages virtual resources of a slice. SEM also exchanges information of virtual resources with other slice element managers via a dedicated resource interface. SEM provides also capability exposure facilities by allowing 3rd parties to access / use via APIs information regarding services provided by the slice (e.g. connectivity information, QoS, mobility, autonomicity, etc.) and to dynamically customize the network characteristics for different diverse use cases (e.g. ultra-low latency, ultra-reliability, value-added services for enterprises, etc.) within the limits set of functions by the operator.

Physical Element Manager (PEM) is installed for each control domain. Control domain is defined according to geographic location and control functions. PEM exchanges information of virtual resource with SEM via virtual resource interface and interconverts between virtual resource and physical resource. The PEM orders physical functions (ex. switches) to allocate physical resource via physical resource interface.

Figure 1 shows the high level view of an Autonomic Slice Networking.

It consists of a number of autonomic nodes resources, which interact directly with each other. Those autonomic nodes resources provide a common set of capabilities across a network slice, called the "Autonomic Slice Networking Infrastructure" (ASNI).

The ASN provides functions like naming, addressing, negotiation, synchronization, discovery and messaging.

Autonomic network functions typically span several slices in the network. The atomic entities of an autonomic function are called the "Autonomic Service Agents" (ASA), which are instantiated on slices.

In a horizontal view, autonomic functions span across the network, as well as the Autonomic Slice Networking Infrastructure. In a vertical view, a slice always implements the ASNI, plus it may have one or several Autonomic Service Agents as part of slice capability exposure. The Autonomic Networking Infrastructure (ASNI) therefore is the foundation for autonomic functions. The current charter of the ANIMA WG includes the specification of the ASNI, using a few autonomic functions as use cases. ASNI would represent a customized and an approach [I-D.ietf-anima-reference-model] for implementing a general purposed ASI.

```
     +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
     :            :      Autonomic Slice Function 1   :             :
     : SSA 1      :      SSA 1      :     SSA 1       :      SSA 1  :
     +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
                  :                 :                 :
                  :   +- - - - - - - - - - - - - - +  :
                  :   : Autonomic Slice Function 2 :  :
                  :   :  ASC 2      :     ASC 2    :  :
                  :   +- - - - - - - - - - - - - - +  :
                  :                 :                 :
     +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
     :           Autonomic Slice Networking Infrastructure         :
     +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
     +                                                             +
     +         +-----------------------------------------+         +
     +         | Autonomic Inter-Slice Orchestration     |         +
     +         +-----------------------------------------+         +
     +          |              |                        |          +
     +----------+       +-----------+                   +----------+
     |Slice 1   |       |Slice 2    |                   | Slice N  |
     |  SEM     |-------|  SEM      |------ ...   ----  |  SEM     |
     |          |       |           |                   |          |
     +----------+       +-----------+                   +----------+
           |                  |                              |
```

```
        +--------------------------------------------------------------+
        |                                                              |
        |    PEC1              PEC2                           PECm     |
        |     |  ...           |         ...                   |       |
        |                                                           |
        |          Resources / Network Functions / ANI                 |
        |                                                              |
        +--------------------------------------------------------------+
             |            |                  |                    |
        +----------------------+    +------------+      +------------+
        +-------+    +---------+    +  +-------+ +      + +--------+ +
        | Node1.1 --| Node1.N |------ |Node2.x|-...------ | NodeM.y| +
        +-------+    +---------+    +  +-------+ +      + +--------+ +
        +----------------------+    +------------+      +------------+
               Domain 1                 Domain 2           Domain M
```

Figure 1: High level view of Autonomic Slice Networking

Additionally, at least 2 autonomous functions are envisioned - Autonomous Slice control (ASC) and Slice Service agent (SSA).  These are explained in sections below.

## 4.  Autonomic Inter-Slice Orchestration

This section describes an autonomic orchestration and its functionality.

Orchestration refers to the system functions that:

* automated and autonomically  co-ordination of network functions in slices

* autonomically coordinate the slices lifecycle and all the components that are part of the slice (i.e.  Service Instances, Network Slice Instances, Resources, Capabilities exposure) to ensure an optimized allocation of the necessary resources across the network.

* coordinate a number of interrelated resources, often distributed across a number of subordinate domains, and to assure transactional integrity as part of the process [TETT1].

* autonomically control of slice life cycle management, including concatenation of slices in each segment of the infrastructure including  the data pane, the control plane, and the management plane.

* autonomically coordinate and trigger of slice elasticity and placement of logical resources in slices.

* coordinates and (re)-configure logical resources in the slice by taking over the control of all the virtualized network functions assigned to the slice.

It is also the continuing process of allocating resources to satisfy contending demands in an optimal manner [TETT2]. The idea of optimal would include at least prioritized SLA commitments [SERMODEL], and factors such as customer endpoint location, geographic or topological proximity, delay, aggregate or fine-grained load, monetary cost, fate- sharing or affinity. The word continuing incorporates recognition that the environment and the service demands constantly change over the course of time, so that orchestration is a continuous, multi-dimensional optimization feedback loop [I-D.strassner-anima-control-loops].

It protects the infrastructure from instabilities and side effects due to the presence of many slice components running in parallel. It ensures the proper triggering sequence of slice functionality and their stable operation. It defines conditions/constraints under which service components will be activated, taking into account operator service and network requirements (inclusive of optimize the use of the available network & compute resources and avoid situations that can lead to sub-par performance and even unstable and oscillatory behaviors.

## 5. GRASP Resource Reservation / Release Messages flow

```
   Inter           Slice           Physical
   Slice           Element         Element           Domain      Physical
   Orchestrator    Manager         Manager           Manager     Function


   |                  |               |                 |               |
   | GRASP Discovery  |GRASP Discovery|GRASP Discovery  |GRASP Discovery|
   |   -Response      |    -Response  |  -Response      | -Response     |
   |  <-------------->| <------------>| <----------->   | <-----------> |
   |                  |               |                 |               |
   | GRASP Request    |               |                 |               |
   |Slicing Objective | GRASP Request |                 |               |
   | ------------->   | Slicing       |                 |               |
   |                  | Objectives    | GRASP Request   |               |
   |                  | ------------> | Slicing         |GRASP Request  |
   |                  |               | Objectives      |Slicing        |
```

```
   |                  |               | ----------->    |Objectives     |
   |                  |               |                 |----------->    |
   |                  |               | GRASP           |               |
   |                  |               | Confirm-Waiting |               |
   |                  |               |  <---------     |               |
   |                  |GRASP          |                 |               |
   |                  |Confirm-Waiting|                 |GRASP          |
   |                  | <-----------  |                 |Negotiation    |
   |                  |               |                 |Single/Multiple|
   |                  |               |GRASP Negotiation|Rounds         |
   |                  |               |Single/Multiple  |<-----------> |
   |                  |               |Rounds           |               |
   | GRASP            |               | <----------->   |               |
   | Confirm-Waiting  |               |                 |               |
   |<---------------  |GRASP          |                 |               |
   |                  |Negotiation    |                 |               |
   |                  |Single/Multiple|                 |               |
   |                  |Rounds         |                 |               |
   |GRASP Negotiation | <-----------> |                 |               |
   |Single/Multiple   |               |                 |               |
   |Rounds            |               |                 |               |
   | <------------>   |               |                 |               |
```

Figure 2 - GRASP: Network Slice reservation / Release3 Messages Flow

The above message sequence figure shows the message flows of the interactions between Inter-Slice Orchestrator, Slice Element Manager, Physical Element Manager, Domain Manager and Physical Network functions.

## 6. The Autonomic Network Slicing Element

This section describes an autonomic slice network element and its internal architecture. The reference model explained in the document "Autonomic Networking - Definitions and Design Goals" [RFC7575] shows the sources of information that an autonomic service agent can leverage: Self-management, Self-knowledge, network knowledge (through discovery), Intent [I-D.du-anima-an-intent], and feedback loops. Fundamentally, there are two levels inside an autonomic node: the level of Autonomic Service Agents, and the level of the Autonomic Slice Networking Infrastructure, with the former using the services of the latter. The self management functionality (self-configuration, self-optimisation, self- healing) could be implemented across the Inter Slice Orchestrator, Slice Element Manager and Physical Element Manager. Such functionality deals with dynamic

* coordination the life cycle of slices

* allocation of resources to slice instances in an efficient way that provides required slice instances performance,

* self-configuration, self-optimization and self-healing of slice instances during their lifecycle management including deployment and operations

* self-configuration, self-optimization and self-healing of services of each slice instance. Service lifecycle, that is typically different than slice instance lifecycle should also be managed in the autonomous way.

Figure 3 illustrates this concept.

```
+------------------------------------------------------------+
|                                                            |
| +-----------+        +------------+        +------------+  |
| | Autonomic |        | Autonomic  |        | Autonomic  |  |
| | Service   |        | Service    |        | Service    |  |
| | Agent 1   |        | Agent 2    |        | Agent 3    |  |
| +-----------+        +------------+        +------------+  |
|       ^                    ^                     ^         |
|  - - -| - - API level - - | - - - - - - - - - - |- - - - - |
|       V                    V                     V        |
|------------------------------------------------------------|
| Autonomic Slice Networking Infrastructure                  |
|    - Service characteristics (ultra-low latency,           |
|      ultra-reliability, etc)                               |
|    - Autonomic Control Plane functions                     |
|    - Autonomic Management Plane functions                  |
|    - Self-x functions and related control loops elements   |
|    - Autonomic Slice Addressing                            |
|      Discovery, negotiation and synchronisation functions  |
|    - Intent distribution                                   |
|    - Aggregated reporting and feedback loops               |
|    - Routing                                               |
|    - Security mechanisms                                   |
|------------------------------------------------------------|
|             Basic Operating System Functions               |
+------------------------------------------------------------+
```

Figure 3: Model of an autonomic element

The Autonomic Slice Networking Infrastructure (lower part of Figure 2) contains slice specific data structures, for example trust information about itself and its peers, as well as a generic set of functions, independent of a particular usage. This infrastructure should be generic, and support a variety of Autonomic Service Agents (upper part of Figure 2). The Autonomic Control Plane is the summary of all interactions of the Autonomic Slice Networking Infrastructure with other services.

The use cases of "Autonomics" such as self-management, self-optimisation, etc, are implemented as Autonomic Service Agents. They use the services and data structures of the underlying autonomic networking infrastructure. The Autonomic Slice Networking Infrastructure should itself be self-managing.

The "Basic Operating System Functions" include the "normal OS", including the network stack, security functions, etc. Autonomic Network Slicing Element is a composition of autonomic slice service agents and autonomic slice control. Autonomic slice service agents obtain specific network resources and provide self-managing and self-controlling functions. An autonomic slice control is a higher-level autonomic function that takes the role of life-cycle management of a or many slice instances. There can be many slice control functions based on different types or attributes of slice.

## 7. The Autonomic Slice Networking Ianfrastructure

The Autonomic Networking Infrastructure provides a layer of common functionality across an Autonomic Network. It comprises "must implement" functions and services, as well as extensions. The Autonomic Slice Networking Infrastructure (ASNI) resides on top of an abstraction layer of resource, network function and network infrastructure as shown in figure 1. The document assumes abstraction layer enables different autonomous service agents to communicate with the underlying disaggregated and distributed network infrastructure, which itself maybe an autonomous networking (AN) domain or combination of multiple AN domain. The goal of ASNI is to provide autonomic life-cycle management of network slices.

### 7.1. Signaling Between Autonomic Slice Element Managers

The basic network capabilities are autonomically or through traditional techniques are learnt by slice agents. This depends on the fact that physical infrastructure is an autonomic network or not. The GASP extensions signaling [I-D.liu-anima-grasp-distribution] [I-D.liu-anima-grasp-api] [I-D.ietf-anima-grasp] may be used for

* Discovery of SEMs - a process by which an one SEM discovers peers according to a specific discovery objective. The discovered SEMs peers may later be used as negotiation counterparts or as sources of other coordination activities.

* Negotiation between SEMs - a process by which two SEMs interact to agree on slice logical resource settings that best satisfy the objectives of both SEMs.

* The Synchronization between SEMs - a process by which Orchestrator and SEMs interact to receive the current state of capability exposure values used at a given time in other SEM. This is a special case of negotiation in which information is sent but the SEM or Orchestrator do not request their peers to change configuration settings.

* Self configuration of SEMs - a process by which Orchestrator and SEMs interact to receive the current state of capability exposure values used at a given time in other SEM. This is a special case of synchronization in which information is sent and the SEM is requesting their peers to change configuration settings.

* Self optimization of SEMs - a process by which Orchestrator and SEMs interact to receive the current state of capability exposure values used at a given time in other SEMs. This is a special case of configuration in which information is sent and the SEM is requesting their peers to change logical resource settings in a slice based on an optimisation criteria.

* Mediation for slice resources - a process by which two SEMs interact to agree to logically move resources between slices that best satisfy the objectives of both SEMs triggering of slice elasticity and placement of logical resources in slices. Th???is is a special case of negotiation in which information is sent Orchestrator do request SEMs to change logical resource configuration settings.

* Triggering and governing of elasticity ? a process for autonomic scaling intent configuration mechanisms and resources on the slice level; it allows rapid provisioning, automatic scaling out, or in, of resources. Scale in/out criteria might be used for network autonomics in order the controller to react to a certain set of variations in monitored slices.

* Providing on-demand a self-service network slicing.

Optionally, SSA capabilities are more interesting to slice control autonomic functions for slice creation and install. The slice control must have the independent intelligence to process and filter capabilities to meet a network slice specification and have low level resources allocated for a slice through SSAs.

## 7.2. The Autonomic Control Plane

TBD.

## 7.3. Naming & Addressing

A slice can be instantiated on demand, represents a logical network and therefore, must be assigned a unique identifier. A Slice Service Agent (SSA) may support functions of a single or multiple slices and communicate with each other, using the addressing of the Autonomic or traditional (non-autonomic) Networking Infrastructure reside on. An

SSA complies with ACP addressing mechanisms and in a domain, i.e., As part of the enrolment process the registrar assigns a number to the device, which is unique for slicing registrar and in ASNI domain.

## 7.4. Discovery

Slices themselves are not discovered but are instantiated through slice control autonomic function. However, both slice service agents and slice control functions must be discovered. Even though autonomic control plane will support discovery of all the SSAs and slice control, it may not be necessary.

## 7.5. Routing

Autonomic network slicing follows single routing protocol as described in [I-D.ietf-anima-autonomic-control-plane].

# 8. Security and Trust Infrastructure

An Autonomic Slice Network is self-protecting. All protocols are secure by default, without the requirement for the administrator to explicitly configure security.

TBD.

## 8.1. Public Key Infrastructure

An autonomic domain uses a PKI model. The root of trust is a certification authority (CA). A registrar acts as a registration authority (RA).

A minimum implementation of an autonomic domain contains one CA, one Registrar, and network elements.

## 8.2. Domain Certificate

TBD.

## 9. Cross-Domain Functionality

TBD.

## 10. Autonomic Service Agents (ASA)

This section describes how autonomic services run on top of the Autonomic Slice Networking Infrastructure. There are at least two different types of autonomic functions are known:

1. Slice Service Agents are low level functions that learn capabilities of underlying infrastructure in terms of interfaces and available resources. They coordinate with Slice control to associate these resources with specific slice instances in effect performing full life cycle management of these resources.
2. Slice Control Autonomic Function: Slice control is responsible for high-level life-cycle management of a slice itself. This function will hold slice instances and their attributes related data structures in autonomic network slice infrastructure. As an example, a slice is defined for high bandwidth, highly secure transactional application. A slice control must be capable of negotiating resources required across different SSAs.

Out of scope are details of the mechanisms how the information is represented and exchanged between the two autonomic functions.

## 11. Management and Programmability

This section describes how an Autonomic Network is managed, and programmed.

### 11.1. How a Slice Network Is Managed

Slice autonomic management is driven by Slice Element Managers, there are five categories operation:

1. Creating a network slice: Receive a network slice resource description request, upon successful negotiation with SSA allocate resource for it.
2. Shrink/Expand slice network: Dynamically alter resource requirements for a running slice network according service load.
3. (Re-)Configure slice network: The slice management user deploys a user level service into the slice. The slice control takes over the control of all the virtualized network functions and network programmability functions assigned to the slice, and

(re-)configure them as appropriate to provide the end-to-end service.

5. Self-X slice operation: namely self-configuration, self-composition, self-monitoring, self-optimisation, self-elasticity would be carried out as part of new slice protocols.

### 11.2. Autonomic Resource Information Model

TBD.

The proposed autonomic resource information model is presented as a tree structure of attributes including the following elements: connectivity resources, storage resources, compute resources, service instances, network slice level attributes, etc. The Yang language would be used to represent the autonomic resource information model.

### 11.3. Control Loops

TBD.

### 11.4. APIs

The API model of for autonomic slicing semantically, is grouped into the following APIs to be defined.

#### 11.4.1. Slice Control APIs

1. Create a slice network on user request. The request includes resource description. A unique identify a slice network, group all the resource.
2. Destroy a slice network identified by it’s id.
3. Query a slice network slicing state by it’s uuid.
4. Modify a slice network.

#### 11.4.2. Service Agent - Device APIs

A service agent will interface with the physical infrastructure either through an autonomic network or traditional infrastructure. Depending upon which a device can either have autonomic or non-autonomic addressing. Service agents are required to perform life cycle management of network elements participating in a network slice and the following APIs are needed for addition, removal or update of a specific device. A device may be a logical or physical network element. Optionally, it may be a network function.

#### 11.4.3. Service Agent - Port APIs

A port may be a physical or logical network port in a slice depending upon whether underlying infrastructure is an autonomic or traditional network. Service agents must be able to control the operational state of these ports. APIs are needed for addition, removal, update and operational state retrieval of a specific port.

#### 11.4.4. Service Agent - Link APIs

A link connects two or more ports of devices described in above section. Service agents must be able to control the operational and connection status of these links through APIs for addition, removal, update and state retrieval for each link.

### 11.5. Relationship with MANO

Please refer to [MANO] for MANO introduction.

## 12. Security Considerations

### 12.1. Threat Analysis

TBD.

### 12.2. Security Mechanisms

TBD.

## 13. IANA Considerations

This document requests no action by IANA.

## 14. Acknowledgements

This document was converted to nroff by Stuart Clayman (UCL) to comply with RFC format [RFC2629].

## 14. References

### 14.1. Normative References

[I-D.ietf-anima-grasp] Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", draft-ietf-anima- grasp-10 (work in progress), March 2017.

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

[RFC7665] Halpern, J., Pignataro, C., "Service Function Chaining (SFC) Architecture", October 2015 <https://tools.ietf.org/html/rfc7665>.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <http://www.rfc-editor.org/info/rfc2629>.

14.2. Informative References

[ChinaCom09] A. Galis et all - "Management and Service-aware Networking Architectures (MANA) for Future Internet" - Invited paper IEEE 2009 Fourth International Conference on Communications and Networking in China (ChinaCom09) 26-28 August 2009, Xi’an, China, <http://www.chinacom.org/2009/index.html>.

[GENI] "GENI Key Concepts - Global Environment for Network Innovations (GENI)" <http://groups.geni.net/geni/wiki/GENIConcepts>.

[I-D.du-anima-an-intent] Du, Z., Jiang, S., Nobre, J., Ciavaglia, L., and M. Behringer, "ANIMA Intent Policy and Format", draft-du- anima-an-intent-04 (work in progress), July 2016.

[I-D.ietf-anima-autonomic-control-plane] Behringer, M., Eckert, T., and S. Bjarnason, "An Autonomic Control Plane", draft-ietf-anima-autonomic-control- plane-03 (work in progress), July 2016.

[I-D.ietf-anima-reference-model] Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., Pierre, P., Liu, B., Nobre, J., and J. Strassner, "A Reference Model for Autonomic Networking", draft-ietf- anima-reference-model-02 (work in progress), July 2016.

[I-D.liu-anima-grasp-api] Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)", draft-liu-anima-grasp-api-02 (work in progress), September 2016.

[I-D.liu-anima-grasp-distribution] Liu, B. and S. Jiang, "Information Distribution over GRASP", draft-liu-anima-grasp-distribution-02 (work in progress), September 2016.

[I-D.strassner-anima-control-loops] Strassner, J., Halpern, J., and M. Behringer, "The Use of Control Loops in Autonomic Networking", draft-strassner- anima-control-loops-01 (work in progress), April 2016.

[IMT2020] ITU-T IMT2020 document "Report on Gap Analysis" - ITU-T IMT2020 ITU- Dec 2015 Published by ITU-T IMT2020. <http://www.itu.int/en/ITU-T/focusgroups/imt-2020/Pages/default.aspx>.

[MANO] "ETSI European Telecommunications Standards Institute. Network Functions Virtualisation (NFV); Management and Orchestration v1.1.1." Website, December 2014. <http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_ nfv-man001v010101p.pdf>.

[NGMN] Hedmar,P., Mschner, K., et all - NGMN Alliance document "Description of Network Slicing Concept", January 2016. <https://www.ngmn.org/uploads/media/160113_Network_Slicing_v1_0.pdf>.

[NGS-3GPP] "Study on Architecture for Next Generation System" - latest version v1.0.2 September 2016 <http://www.3gpp.org/ftp/tsg_sa/WG2_Arch/Latest_SA2_Specs/Latest_draft_S2_Specs>.

[ONF] Paul, M, Schallen, S., Betts, M., Hood, D., Shirazipor, M., Lopes, D., Kaippallimalit, J., - Open Network Fundation document "Applying SDN Architecture to 5G Slicing", April 2016. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Applying_SDN_Architecture_to_5G_Slicing_TR-526.pdf>.

[NS1] L. Geng, J. Dong, S. Bryant, K., Makhijani, A., Galis, X. de Foy, S. Kuklinski, - "Network Slicing Architecture", July 2017. <https://tools.ietf.org/html/draft-geng-netslices- architecture-02>.

[NS2] L. Geng, L. Wang, S. Kuklinski, L. Qiang, S. Matsushima, A., Galis, L. Contreras - "Problem Statement of Supervised Heterogeneous Network Slicing", October 2017 <https://datatracker.ietf.org/doc/draft-geng-coms-problem-statement/>.

[ASN] A., Galis, K., Makhijani, D. Yu, B. Liu - "Autonomic Slice Networking-Requirements and Reference Model" - May 2017 < https://datatracker.ietf.org/doc/draft-galis-anima-autonomic-slice-networking/>.

[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, <http://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, July 2016, <http://www.rfc-editor.org/info/rfc7576>.

[TETT1] Guerzoni, R., Vaishnavi, I., Pares-Caparros, D., Galis, A., et al, "Analysis of End-to-End Multi Domain Management and Orchestration Frameworks for Software Defined Infrastructures: an Architectural Survey", Transactions on Emerging Telecommunications Technologies, Wiley Online Library, DOI: 10.1002/ett.3103, June 2016, <onlinelibrary.wiley.com/doi/10.1002/ett.3103/pdf>.

[TETT2] Karl, H., Draxler, S., Peuster, M, Galis, A., et all "DevOps for Network Function Virtualization: An Architectural Approach", Transactions on Emerging Telecommunications Technologies Wiley Online Library, DOI: 10.1002/ett.3084, July 2016, <http://onlinelibrary.wiley.com/doi/10.1002/ett.3084/full>.

[SERMODEL] C., Borman, B. Carpenter, B., Liu, "Service Models Explained " draft-wu-opsawg-service-model-explained-05 <https://datatracker.ietf.org/doc/draft-wu-opsawg-service-model- explained/>.

[5GNS] Galis, A. (UCL), Chih-Lin I (China Mobile) - "Towards 5G Network Slicing - Motivations and Challenges" March 2017, IEEE 5G Tech Focus, Volume 1, Number 1, March 2017- <http://5g.ieee.org/tech-focus/march-2017#networkslicing>.

[PER-NS] Galis, A. - " Perspectives on Network Slicing - Towards the New 'Bread and Butter' of Networking and Servicing", IEEE SDN Initiative - January 2018 <https://sdn.ieee.org/newsletter/january-2018/perspectives-on- network-slicing-towards-the-new-bread-and-butter-of-networking-and-servicing>.

[NS-ETSI] "Network Functions Virtualisation (NFV) Release 3; Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architecture Framework- ETSI GR NFV-EVE 012 V3.1.1 (2017-12)" <http://www.etsi.org/deliver/etsi_gr/NFV-

EVE/001_099/012/03.01.01_60/gr_NFV-EVE012v030101p.pdf>

Authors' Addresses

Alex Galis (editor)
University College London
Department of Electronic and Electrical Engineering
Torrington Place
London WC1E 7JE
United Kingdom

Email: a.galis@ucl.ac.uk

Kiran Makhijani
Huawei Technologies
2890, Central Expressway
Santa Clara CA 95032
USA

Email: USA Email: kiran.makhijani@huawei.com

Delei Yu
Huawei Technologies
Q22, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: yudelei@huawei.com

Bing Liu
Huawei Technologies Co., Ltd
Q14, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: leo.liubing@huawei.com

ANIMA WG
Internet-Draft
Intended status: Standards Track
Expires: 3 May 2021

T. Eckert, Ed.
Futurewei USA
M. Behringer, Ed.

S. Bjarnason
Arbor Networks

30 October 2020

# An Autonomic Control Plane (ACP)
## draft-ietf-anima-autonomic-control-plane-30

## Abstract

Autonomic functions need a control plane to communicate, which depends on some addressing and routing. This Autonomic Control Plane should ideally be self-managing, and as independent as possible of configuration. This document defines such a plane and calls it the "Autonomic Control Plane", with the primary use as a control plane for autonomic functions. It also serves as a "virtual out-of-band channel" for Operations, Administration and Management (OAM) communications over a network that provides automatically configured hop-by-hop authenticated and encrypted communications via automatically configured IPv6 even when the network is not configured, or misconfigured.

## Status of This Memo



## Copyright Notice

Table of Contents

## 1. Introduction (Informative)

Autonomic Networking is a concept of self-management: Autonomic functions self-configure, and negotiate parameters and settings across the network. [RFC7575] defines the fundamental ideas and design goals of Autonomic Networking. A gap analysis of Autonomic Networking is given in [RFC7576]. The reference architecture for Autonomic Networking in the IETF is specified in the document [I-D.ietf-anima-reference-model].

Autonomic functions need an autonomically built communications infrastructure. This infrastructure needs to be secure, resilient and re-usable by all autonomic functions. Section 5 of [RFC7575] introduces that infrastructure and calls it the Autonomic Control Plane (ACP). More descriptively it would be the "Autonomic communications infrastructure for OAM and Control". For naming consistency with that prior document, this document continues to use the name ACP though.

Today, the OAM and control plane of IP networks is what is typically called in-band management/signaling: Its management and control protocol traffic depends on the routing and forwarding tables, security, policy, QoS and potentially other configuration that first has to be established through the very same management and control protocols. Misconfigurations including unexpected side effects or mutual dependences can disrupt OAM and control operations and especially disrupt remote management access to the affected node itself and potentially a much larger number of nodes for whom the affected node is on the network path.

For an example of inband management failing in the face of operator induced misconfiguration, see [FCC], for example III.B.15 on page 8: "...engineers almost immediately recognized that they had misdiagnosed the problem. However, they were unable to resolve the issue by restoring the link because the network management tools required to do so remotely relied on the same paths they had just disabled".

Traditionally, physically separate, so-called out-of-band (management) networks have been used to avoid these problems or at least to allow recovery from such problems. Worst case, personnel are sent on site to access devices through out-of-band management ports (also called craft ports, serial console, management ethernet port). However, both options are expensive.

In increasingly automated networks either centralized management systems or distributed autonomic service agents in the network require a control plane which is independent of the configuration of the network they manage, to avoid impacting their own operations through the configuration actions they take.

This document describes a modular design for a self-forming, self-managing and self-protecting ACP, which is a virtual out-of-band network designed to be as independent as possible of configuration, addressing and routing to avoid the self-dependency problems of current IP networks while still operating in-band on the same physical network that it is controlling and managing. The ACP design is therefore intended to combine as well as possible the resilience of out-of-band management networks with the low-cost of traditional IP in-band network management. The details how this is achieved are described in Section 6.

In a fully autonomic network node without legacy control or management functions/protocols, the Data-Plane would be for example just a forwarding plane for "Data" IPv6 packets, aka: packets other than the control and management plane packets that are forwarded by the ACP itself. In such networks/nodes, there would be no non-autonomous control or non-autonomous management plane.

Routing protocols for example would be built inside the ACP as so-called autonomous functions via autonomous service agents, leveraging the ACP's functions instead of implementing them separately for each protocol: discovery, automatically established authenticated and encrypted local and distant peer connectivity for control and management traffic, and common control/management protocol session and presentation functions.

When ACP functionality is added to nodes that have non-autonomous management plane and/or control plane functions (henceforth called non-autonomous nodes), the ACP instead is best abstracted as a special Virtual Routing and Forwarding (VRF) instance (or virtual router) and the complete pre-existing non-autonomous management and/or control plane is considered to be part of the Data-Plane to avoid introduction of more complex, new terminology only for this case.

Like the forwarding plane for "Data" packets, the non-autonomous control and management plane functions can then be managed/used via the ACP. This terminology is consistent with pre-existing documents such as [RFC8368].

In both instances (autonomous and non-autonomous nodes), the ACP is built such that it is operating in the absence of the Data-Plane, and in the case of existing non-autonomous (management, control) components in the Data-Plane also in the presence of any (mis-)configuration thereof.

The Autonomic Control Plane serves several purposes at the same time:

1. Autonomic functions communicate over the ACP. The ACP therefore directly supports Autonomic Networking functions, as described in [I-D.ietf-anima-reference-model]. For example, Generic Autonomic Signaling Protocol (GRASP - [I-D.ietf-anima-grasp]) runs securely inside the ACP and depends on the ACP as its "security and transport substrate".
2. A controller or network management system can use it to securely bootstrap network devices in remote locations, even if the (Data-Plane) network in between is not yet configured; no Data-Plane dependent bootstrap configuration is required. An example of such a secure bootstrap process is described in [I-D.ietf-anima-bootstrapping-keyinfra].
3. An operator can use it to access remote devices using protocols such as Secure SHell (SSH) or Network Configuration Protocol (NETCONF) running across the ACP, even if the network is misconfigured or not configured.

This document describes these purposes as use cases for the ACP in Section 3, it defines the requirements in Section 4. Section 5 gives an overview of how the ACP is constructed.

The normative part of this document starts with Section 6, where the ACP is specified. Section 7 explains how to support ACP on L2 switches (normative). Section 8 explains how non-ACP nodes and networks can be integrated (normative).

The remaining sections are non-normative: Section 10 reviews benefits of the ACP (after all the details have been defined), Section 9 provides operational recommendations, Appendix A provides additional explanations and describes additional details or future standard or proprietary extensions that were considered not to be appropriate for standardization in this document but were considered important to document. There are no dependencies against Appendix A to build a complete working and interoperable ACP according to this document.

The ACP provides secure IPv6 connectivity, therefore it can be used not only as the secure connectivity for self-management as required for the ACP in [RFC7575], but it can also be used as the secure connectivity for traditional (centralized) management. The ACP can be implemented and operated without any other components of autonomic networks, except for the GRASP protocol. ACP relies on per-link DULL GRASP (see Section 6.4) to autodiscover ACP neighbors, and includes the ACP GRASP instance to provide service discovery for clients of the ACP (see Section 6.9) including for its own maintenance of ACP certificates.

The document "Using Autonomic Control Plane for Stable Connectivity of Network OAM" [RFC8368] describes how the ACP alone can be used to provide secure and stable connectivity for autonomic and non-autonomic OAM applications, specifically for the case of current non-autonomic networks/nodes. That document also explains how existing management solutions can leverage the ACP in parallel with traditional management models, when to use the ACP and how to integrate with potentially IPv4 only OAM backends.

Combining ACP with Bootstrapping Remote Secure Key Infrastructures (BRSKI), see [I-D.ietf-anima-bootstrapping-keyinfra]) results in the "Autonomic Network Infrastructure" (ANI) as defined in [I-D.ietf-anima-reference-model], which provides autonomic connectivity (from ACP) with secure zero-touch (automated) bootstrap from BRSKI. The ANI itself does not constitute an Autonomic Network, but it allows the building of more or less autonomic networks on top of it - using either centralized, Software Defined Networking-(SDN-)style (see [RFC7426]) automation or distributed automation via Autonomic Service Agents (ASA) / Autonomic Functions (AF) - or a mixture of both. See [I-D.ietf-anima-reference-model] for more information.

## 1.1. Applicability and Scope

Please see the following Terminology section (Section 2) for explanations of terms used in this section.

The design of the ACP as defined in this document is considered to be applicable to all types of "professionally managed" networks: Service Provider, Local Area Network (LAN), Metro(politan networks), Wide Area Network (WAN), Enterprise Information Technology (IT) and ->"Operational Technology" (OT) networks. The ACP can operate equally on layer 3 equipment and on layer 2 equipment such as bridges (see Section 7). The hop-by-hop authentication, integrity-protection and confidentiality mechanism used by the ACP is defined to be negotiable, therefore it can be extended to environments with different protocol preferences. The minimum implementation requirements in this document attempt to achieve maximum interoperability by requiring support for multiple options depending on the type of device: IPsec, see [RFC4301], and Datagram Transport Layer Security (DTLS, see Section 6.8.4).

The implementation footprint of the ACP consists of Public Key Infrastructure (PKI) code for the ACP certificate including "Enrollment over Secure Transport (EST, see [RFC7030]), the GRASP protocol, UDP, TCP and Transport Layer Security (TLS, see Section 6.1), for security and reliability of GRASP and for EST, the ACP secure channel protocol used (such as IPsec or DTLS), and an instance of IPv6 packet forwarding and routing via the Routing Protocol for Low-power and Lossy Networks (RPL), see [RFC6550], that is separate from routing and forwarding for the Data-Plane (user traffic).

The ACP uses only IPv6 to avoid complexity of dual-stack ACP operations (IPv6/IPv4).  Nevertheless, it can without any changes be integrated into even otherwise IPv4-only network devices.  The Data-Plane itself would not need to change and it could continue to be IPv4 only.  For such IPv4-only devices, the IPv6 protocol itself would be additional implementation footprint that is only required for the ACP.

The protocol choices of the ACP are primarily based on wide use and support in networks and devices, well understood security properties and required scalability.  The ACP design is an attempt to produce the lowest risk combination of existing technologies and protocols to build a widely applicable operational network management solution.

RPL was chosen because it requires a smaller routing table footprint in large networks compared to other routing protocols with an autonomically configured single area.  The deployment experience of large scale Internet of Things (IoT) networks serves as the basis for wide deployment experience with RPL.  The profile chosen for RPL in the ACP does not leverage any RPL specific forwarding plane features (IPv6 extension headers), making its implementation a pure control plane software requirement.

GRASP is the only completely novel protocol used in the ACP, and this choice was necessary because there is no existing suitable protocol to provide the necessary functions to the ACP, so GRASP was developed to fill that gap.

The ACP design can be applicable to devices constrained with respect to cpu and memory, and to networks constrained with respect to bitrate and reliability, but this document does not attempt to define the most constrained type of devices or networks to which the ACP is applicable. RPL and DTLS for ACP secure channels are two protocol choices already making ACP more applicable to constrained environments. Support for constrained devices in this specification is opportunistic, but not complete, because the reliable transport for GRASP (see Section 6.9.2) only specifies TCP/TLS. See Appendix A.8 for discussions about how future standards or proprietary extensions/variations of the ACP could better meet different expectations from those on which the current design is based including supporting constrained devices better.

## 2. Acronyms and Terminology (Informative)

[RFC-Editor: Please add ACP, BRSKI, GRASP, MASA to https://www.rfc-editor.org/materials/abbrev.expansion.txt.]

[RFC-Editor: What is the recommended way to reference a hanging text, e.g. to a definition in the list of definitions? Up to -28, this document was using XMLv2 and the only option I could find for RFC/XML to point to a hanging text was format="title", which leads to references such as '->"ACP certificate" ()', aka: redundant empty parenthesis. Many reviewers where concerned about this. I created a ticket to ask for an xml2rfc enhancement to avoid this in the future: https://trac.tools.ietf.org/tools/xml2rfc/trac/ticket/347. When i changed to XMLv3 in version -29, i could get rid of the unnecessary () by using format="none", but that format is declared to be deprecated in XMLv3. So i am not aware of any working AND "non-deprecated" option.]

[RFC-Editor: Question: Is it possible to change the first occurrences of [RFCxxxx] references to "rfcxxx title" [RFCxxxx]? the XML2RFC format does not seem to offer such a format, but I did not want to duplicate 50 first references - one reference for title mentioning and one for RFC number.]

This document serves both as a normative specification for how ACP nodes have to behave as well as describing requirements, benefits, architecture and operational aspects to explain the context. Normative sections are labelled "(Normative)" and use BCP 14 keywords. Other sections are labelled "(Informative)" and do not use those normative keywords.

In the rest of the document we will refer to systems using the ACP as "nodes". Typically, such a node is a physical (network equipment) device, but it can equally be some virtualized system. Therefore, we do not refer to them as devices unless the context specifically calls for a physical system.

This document introduces or uses the following terms (sorted alphabetically). Terms introduced are explained on first use, so this list is for reference only.

ACP: "Autonomic Control Plane". The Autonomic Function as defined in this document. It provides secure zero-touch (automated) transitive (network wide) IPv6 connectivity for all nodes in the same ACP domain as well as a GRASP instance running across this ACP IPv6 connectivity. The ACP is primarily meant to be used as a component of the ANI to enable Autonomic Networks but it can equally be used in simple ANI networks (with no other Autonomic Functions) or completely by itself.

ACP address: An IPv6 address assigned to the ACP node. It is stored in the acp-node-name of the ->"ACP certificate".

ACP address range/set: The ACP address may imply a range or set of addresses that the node can assign for different purposes. This address range/set is derived by the node from the format of the ACP address called the "addressing sub-scheme".

ACP connect interface: An interface on an ACP node providing access to the ACP for non ACP capable nodes without using an ACP secure channel. See Section 8.1.1.

ACP domain: The ACP domain is the set of nodes with ->"ACP certificates" that allow them to authenticate each other as members of the ACP domain. See also Section 6.2.3.

ACP (ANI/AN) certificate: A [RFC5280] certificate (LDevID) carrying the acp-node-name which is used by the ACP to learn its address in the ACP and to derive and cryptographically assert its membership in the ACP domain.

ACP acp-node-name field: An information field in the ACP certificate in which the ACP relevant information is encoded: the ACP domain name, the ACP IPv6 address of the node and optional additional role attributes about the node.

ACP Loopback interface: The Loopback interface in the ACP Virtual Routing and Forwarding (VRF) that has the ACP address assigned to it. See Section 6.13.5.1.

ACP network: The ACP network constitutes all the nodes that have access to the ACP. It is the set of active and transitively connected nodes of an ACP domain plus all nodes that get access to the ACP of that domain via ACP edge nodes.

ACP (ULA) prefix(es): The /48 IPv6 address prefixes used across the ACP. In the normal/simple case, the ACP has one ULA prefix, see Section 6.11. The ACP routing table may include multiple ULA prefixes if the "rsub" option is used to create addresses from more than one ULA prefix. See Section 6.2.2. The ACP may also include non-ULA prefixes if those are configured on ACP connect interfaces. See Section 8.1.1.

ACP secure channel: A channel authenticated via ->"ACP certificates" providing integrity protection and confidentiality through encryption. These are established between (normally) adjacent ACP nodes to carry traffic of the ACP VRF securely and isolated from Data-Plane traffic in-band over the same link/path as the Data-Plane.

ACP secure channel protocol: The protocol used to build an ACP secure channel, e.g., Internet Key Exchange Protocol version 2 (IKEv2) with IPsec or Datagram Transport Layer Security (DTLS).

ACP virtual interface: An interface in the ACP VRF mapped to one or more ACP secure channels. See Section 6.13.5.

AN "Autonomic Network": A network according to [I-D.ietf-anima-reference-model]. Its main components are ANI, Autonomic Functions and Intent.

(AN) Domain Name: An FQDN (Fully Qualified Domain Name) in the acp-node-name of the Domain Certificate. See Section 6.2.2.

ANI (nodes/network): "Autonomic Network Infrastructure". The ANI is the infrastructure to enable Autonomic Networks. It includes ACP, BRSKI and GRASP. Every Autonomic Network includes the ANI, but not every ANI network needs to include autonomic functions beyond the ANI (nor Intent). An ANI network without further autonomic functions can for example support secure zero-touch (automated) bootstrap and stable connectivity for SDN networks - see [RFC8368].

ANIMA: "Autonomic Networking Integrated Model and Approach". ACP, BRSKI and GRASP are specifications of the IETF ANIMA working group.

ASA: "Autonomic Service Agent". Autonomic software modules running on an ANI device. The components making up the ANI (BRSKI, ACP, GRASP) are also described as ASAs.

Autonomic Function: A function/service in an Autonomic Network (AN) composed of one or more ASA across one or more ANI nodes.

BRSKI: "Bootstrapping Remote Secure Key Infrastructures" ([I-D.ietf-anima-bootstrapping-keyinfra]. A protocol extending EST to enable secure zero-touch bootstrap in conjunction with ACP. ANI nodes use ACP, BRSKI and GRASP.

CA: "Certification Authority". An entity that issues digital certificates. A CA uses its private key to sign the certificates it issues. Relying parties use the public key in the CA certificate to validate the signature.

CRL: "Certificate Revocation List". A list of revoked certificates. Required to revoke certificates before their lifetime expires.

Data-Plane: The counterpoint to the ACP VRF in an ACP node: forwarding of user traffic and in non-autonomous nodes/networks also any non-autonomous control and/or management plane functions. In a fully Autonomic Network node, the Data-Plane is managed autonomically via Autonomic Functions and Intent. See Section 1 for more detailed explanations.

device: A physical system, or physical node.

Enrollment: The process through which a node authenticates itself to a network with an initial identity, which is often called IDevID certificate, and acquires from the network a network specific identity, which is often called LDevID certificate, and certificates of one or more Trust Anchor(s). In the ACP, the LDevID certificate is called the ACP certificate.

EST: "Enrollment over Secure Transport" ([RFC7030]). IETF standard-track protocol for enrollment of a node with an LDevID certificate. BRSKI is based on EST.

GRASP: "Generic Autonomic Signaling Protocol". An extensible signaling protocol required by the ACP for ACP neighbor discovery. The ACP also provides the "security and transport substrate" for the "ACP instance of GRASP". This instance of GRASP runs across the ACP secure channels to support BRSKI and other NOC/OAM or Autonomic Functions. See [I-D.ietf-anima-grasp].

IDevID: An "Initial Device IDentity" X.509 certificate installed by the vendor on new equipment. Contains information that establishes the identity of the node in the context of its vendor/manufacturer such as device model/type and serial number. See [AR8021]. The IDevID certificate cannot be used as a node identifier for the ACP because they are not provisioned by the owner of the network, so they can not directly indicate an ACP domain they belong to.

in-band (management/signaling): In-band management traffic and/or control plane signaling uses the same network resources such as routers/switches and network links that it manages/controls. In-band is the standard management and signaling mechanism in IP networks. Compared to ->"out-of-band" it requires no additional physical resources, but introduces potentially circular dependencies for its correct operations. See ->"introduction".

Intent: Policy language of an autonomic network according to [I-D.ietf-anima-reference-model].

Loopback interface: See ->"ACP Loopback interface".

LDevID: A "Local Device IDentity" is an X.509 certificate installed during "enrollment". The Domain Certificate used by the ACP is an LDevID certificate. See [AR8021].

Management: Used in this document as another word for ->"OAM".

MASA (service): "Manufacturer Authorized Signing Authority". A vendor/manufacturer or delegated cloud service on the Internet used as part of the BRSKI protocol.

MIC: "Manufacturer Installed Certificate". This is another word to describe an IDevID in referenced materials. This term is not used in this document.

native interface: Interfaces existing on a node without configuration of the already running node. On physical nodes these are usually physical interfaces; on virtual nodes their equivalent.

NOC: Network Operations Center.

node: A system supporting the ACP according to this document. Can be virtual or physical. Physical nodes are called devices.

Node-ID: The identifier of an ACP node inside that ACP. It is the last 64 (see Section 6.11.3) or 78-bits (see Section 6.11.5) of the ACP address.

OAM: Operations, Administration and Management. Includes Network Monitoring.

Operational Technology (OT): https://en.wikipedia.org/wiki/Operational_Technology: "The hardware and software dedicated to detecting or causing changes in physical processes through direct monitoring and/or control of physical devices such as valves, pumps, etc.". OT networks are today in most cases well separated from Information Technology (IT) networks.

out-of-band (management) network: An out-of-band network is a secondary network used to manage a primary network. The equipment of the primary network is connected to the out-of-band network via dedicated management ports on the primary network equipment. Serial (console) management ports were historically most common, higher end network equipment now also has ethernet ports dedicated only for management. An out-of-band network provides management access to the primary network independent of the configuration state of the primary network. See ->"Introduction"

(virtual) out-of-band network: The ACP can be called a virtual out-of-band network for management and control because it attempts to provide the benefits of a (physical) ->"out-of-band network" even though it is physically carried ->"in-band". See ->"introduction".

root CA: "root Certification Authority". A ->"CA" for which the root CA Key update procedures of [RFC7030], Section 4.4 can be applied.

RPL: "IPv6 Routing Protocol for Low-Power and Lossy Networks". The routing protocol used in the ACP. See [RFC6550].

(ACP/ANI/BRSKI) Registrar: An ACP registrar is an entity (software and/or person) that is orchestrating the enrollment of ACP nodes with the ACP certificate. ANI nodes use BRSKI, so ANI registrars are also called BRSKI registrars. For non-ANI ACP nodes, the registrar mechanisms are undefined by this document. See Section 6.11.7. Renewal and other maintenance (such as revocation) of ACP certificates may be performed by other entities than registrars. EST must be supported for ACP certificate renewal (see Section 6.2.5). BRSKI is an extension of EST, so ANI/BRSKI registrars can easily support ACP domain certificate renewal in addition to initial enrollment.

RPI: "RPL Packet Information". Network extension headers for use with the ->"RPL" routing protocols. Not used with RPL in the ACP. See Section 6.12.1.13.

RPL: "Routing Protocol for Low-Power and Lossy Networks". The routing protocol used in the ACP. See Section 6.12.

sUDI: "secured Unique Device Identifier". This is another word to describe an IDevID in referenced material. This term is not used in this document.

TA: "Trust Anchor". A Trust Anchor is an entity that is trusted for the purpose of certificate validation. Trust Anchor Information such as self-signed certificate(s) of the Trust Anchor is configured into the ACP node as part of Enrollment. See [RFC5280], Section 6.1.1.

UDI: "Unique Device Identifier". In the context of this document unsecured identity information of a node typically consisting of at least device model/type and serial number, often in a vendor specific format. See sUDI and LDevID.

ULA: (Global ID prefix) A "Unique Local Address" (ULA) is an IPv6 address in the block fc00::/7, defined in [RFC4193]. ULA is the IPv6 successor of the IPv4 private address space ([RFC1918]). ULA have important differences over IPv4 private addresses that are beneficial for and exploited by the ACP, such as the Locally Assigned Global ID prefix, which are the first 48-bits of a ULA address [RFC4193], section 3.2.1. In this document this prefix is abbreviated as "ULA prefix".

(ACP) VRF: The ACP is modeled in this document as a "Virtual Routing and Forwarding" instance (VRF). This means that it is based on a "virtual router" consisting of a separate IPv6 forwarding table to which the ACP virtual interfaces are attached and an associated IPv6 routing table separate from the Data-Plane. Unlike the VRFs on MPLS/VPN-PE ([RFC4364]) or LISP XTR ([RFC6830]), the ACP VRF does not have any special "core facing" functionality or routing/ mapping protocols shared across multiple VRFs. In vendor products a VRF such as the ACP-VRF may also be referred to as a so called VRF-lite.

(ACP) Zone: An ACP zone is a set of ACP nodes using the same zone field value in their ACP address according to Section 6.11.3. Zones are a mechanism to support structured addressing of ACP addresses within the same /48-bit ULA prefix.

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.

# 3. Use Cases for an Autonomic Control Plane (Informative)

This section summarizes the use cases that are intended to be supported by an ACP. To understand how these are derived from and relate to the larger set of use cases for autonomic networks, please refer to [RFC8316].

## 3.1. An Infrastructure for Autonomic Functions

Autonomic Functions need a stable infrastructure to run on, and all autonomic functions should use the same infrastructure to minimize the complexity of the network. In this way, there is only need for a single discovery mechanism, a single security mechanism, and single instances of other processes that distributed functions require.

## 3.2. Secure Bootstrap over a not configured Network

Today, bootstrapping a new node typically requires all nodes between a controlling node such as an SDN controller ("Software Defined Networking", see [RFC7426]) and the new node to be completely and correctly addressed, configured and secured. Bootstrapping and configuration of a network happens in rings around the controller - configuring each ring of devices before the next one can be bootstrapped. Without console access (for example through an out-of-band network) it is not possible today to make devices securely reachable before having configured the entire network leading up to them.

With the ACP, secure bootstrap of new devices and whole new networks can happen without requiring any configuration of unconfigured devices along the path: As long as all devices along the path support ACP and a zero-touch bootstrap mechanism such as BRSKI, the ACP across a whole network of unconfigured devices can be brought up without operator/provisioning intervention. The ACP also provides additional security for any bootstrap mechanism, because it can provide encrypted discovery (via ACP GRASP) of registrars or other bootstrap servers by bootstrap proxies connecting to nodes that are to be bootstrapped and the ACP encryption hides the identities of the communicating entities (pledge and registrar), making it more difficult to learn which network node might be attackable. The ACP certificate can also be used to end-to-end encrypt the bootstrap communication between such proxies and server. Note that bootstrapping here includes not only the first step that can be provided by BRSKI (secure keys), but also later stages where configuration is bootstrapped.

## 3.3. Data-Plane Independent Permanent Reachability

Today, most critical control plane protocols and OAM protocols are using the Data-Plane of the network. This leads to often undesirable dependencies between control and OAM plane on one side and the Data-Plane on the other: Only if the forwarding and control plane of the Data-Plane are configured correctly, will the Data-Plane and the OAM/control plane work as expected.

Data-Plane connectivity can be affected by errors and faults, for example misconfigurations that make AAA (Authentication, Authorization and Accounting) servers unreachable or can lock an administrator out of a device; routing or addressing issues can make a device unreachable; shutting down interfaces over which a current management session is running can lock an admin irreversibly out of the device. Traditionally only out-of-band access can help recover from such issues (such as serial console or ethernet management port).

Data-Plane dependencies also affect applications in a Network Operations Center (NOC) such as SDN controller applications: Certain network changes are today hard to implement, because the change itself may affect reachability of the devices. Examples are address or mask changes, routing changes, or security policies. Today such changes require precise hop-by-hop planning.

Note that specific control plane functions for the Data-Plane often want to depend on forwarding of their packets via the Data-Plane: Aliveness and routing protocol signaling packets across the Data-Plane to verify reachability across the Data-Plane, using IPv4 signaling packets for IPv4 routing vs. IPv6 signaling packets for IPv6 routing.

Assuming appropriate implementation (see Section 6.13.2 for more details), the ACP provides reachability that is independent of the Data-Plane. This allows the control plane and OAM plane to operate more robustly:

* For management plane protocols, the ACP provides the functionality of a Virtual out-of-band (VooB) channel, by providing connectivity to all nodes regardless of their Data-Plane configuration, routing and forwarding tables.
* For control plane protocols, the ACP allows their operation even when the Data-Plane is temporarily faulty, or during transitional events, such as routing changes, which may affect the control plane at least temporarily. This is specifically important for autonomic service agents, which could affect Data-Plane connectivity.

The document "Using Autonomic Control Plane for Stable Connectivity of Network OAM" [RFC8368] explains this use case for the ACP in significantly more detail and explains how the ACP can be used in practical network operations.

## 4. Requirements (Informative)

The following requirements were identified for the design of the ACP based on the above use-cases (Section 3). These requirements are informative. The ACP as specified in the normative parts of this document is meeting or exceeding these use-case requirements:

ACP1: The ACP should provide robust connectivity: As far as possible, it should be independent of configured addressing, configuration and routing. Requirements 2 and 3 build on this requirement, but also have value on their own.

ACP2: The ACP must have a separate address space from the Data-Plane. Reason: traceability, debug-ability, separation from Data-Plane, infrastructure security (filtering based on known address space).

ACP3: The ACP must use autonomically managed address space. Reason: easy bootstrap and setup ("autonomic"); robustness (admin cannot break network easily). This document uses Unique Local Addresses (ULA) for this purpose, see [RFC4193].

ACP4: The ACP must be generic, that is it must be usable by all the functions and protocols of the ANI. Clients of the ACP must not be tied to a particular application or transport protocol.

ACP5: The ACP must provide security: Messages coming through the ACP must be authenticated to be from a trusted node, and it is very strongly > recommended that they be encrypted.

Explanation for ACP4: In a fully autonomic network (AN), newly written ASAs could potentially all communicate exclusively via GRASP with each other, and if that was assumed to be the only requirement against the ACP, it would not need to provide IPv6 layer connectivity between nodes, but only GRASP connectivity. Nevertheless, because ACP also intends to support non-AN networks, it is crucial to support IPv6 layer connectivity across the ACP to support any transport and application layer protocols.

The ACP operates hop-by-hop, because this interaction can be built on IPv6 link local addressing, which is autonomic, and has no dependency on configuration (requirement 1). It may be necessary to have ACP connectivity across non-ACP nodes, for example to link ACP nodes over the general Internet. This is possible, but introduces a dependency against stable/resilient routing over the non-ACP hops (see Section 8.2).

# 5. Overview (Informative)

When a node has an ACP certificate (see Section 6.2.1) and is enabled to bring up the ACP (see Section 9.3.5), it will create its ACP without any configuration as follows. For details, see Section 6 and further sections:

1. The node creates a VRF instance, or a similar virtual context for the ACP.
2. The node assigns its ULA IPv6 address (prefix) (see Section 6.11 which is learned from the acp-node-name (see Section 6.2.2) of its ACP certificate (see Section 6.2.1) to an ACP loopback interface (see Section 6.11) and connects this interface into the ACP VRF.
3. The node establishes a list of candidate peer adjacencies and candidate channel types to try for the adjacency. This is automatic for all candidate link-local adjacencies, see Section 6.4 across all native interfaces (see Section 9.3.4). If a candidate peer is discovered via multiple interfaces, this will result in one adjacency per interface. If the ACP node has multiple interfaces connecting to the same subnet across which it is also operating as an L2 switch in the Data-Plane, it employs methods for ACP with L2 switching, see Section 7.
4. For each entry in the candidate adjacency list, the node negotiates a secure tunnel using the candidate channel types. See Section 6.6.
5. The node authenticates the peer node during secure channel setup and authorizes it to become part of the ACP according to Section 6.2.3.
6. Unsuccessful authentication of a candidate peer results in throttled connection retries for as long as the candidate peer is discoverable. See Section 6.7.
7. Each successfully established secure channel is mapped into an ACP virtual interface, which is placed into the ACP VRF. See Section 6.13.5.2.
8. Each node runs a lightweight routing protocol, see Section 6.12, to announce reachability of the ACP loopback address (or prefix) across the ACP.
9. This completes the creation of the ACP with hop-by-hop secure tunnels, auto-addressing and auto-routing. The node is now an ACP node with a running ACP.

Note:

* None of the above operations (except the following explicit configured ones) are reflected in the configuration of the node.
* Non-ACP NMS ("Network Management Systems") or SDN controllers have to be explicitly configured for connection into the ACP.

* Additional candidate peer adjacencies for ACP connections across non-ACP Layer-3 clouds requires explicit configuration. See Section 8.2.

The following figure illustrates the ACP.

```
          ACP node 1                          ACP node 2
       ..................               ..................
secure .                 .   secure      .                 .  secure
channel:  +-----------+  :   channel     :  +-----------+  : channel
..--------| ACP VRF   |---------------------| ACP VRF   |---------..
       : / \         / \   <--routing-->   / \         / \ :
       : \ /         \ /                   \ /         \ / :
..--------| Loopback  |---------------------| Loopback  |---------..
       :  | interface |  :               :  | interface |  :
       :  +-----------+  :               :  +-----------+  :
       :                 :               :                 :
       :   Data-Plane    :...............:   Data-Plane    :
       :                 :    link       :                 :
       :.................:               :.................:
```

Figure 1: ACP VRF and secure channels

The resulting overlay network is normally based exclusively on hop-by-hop tunnels. This is because addressing used on links is IPv6 link local addressing, which does not require any prior set-up. In this way the ACP can be built even if there is no configuration on the node, or if the Data-Plane has issues such as addressing or routing problems.

## 6. Self-Creation of an Autonomic Control Plane (ACP) (Normative)

This section specifies the components and steps to set up an ACP. The ACP is automatically "self-creating", which makes it "indestructible" against most changes to the Data-Plane, including misconfigurations of routing, addressing, NAT, firewall or any other traffic policy filters that inadvertently or otherwise unavoidably would also impact the management plane traffic, such as the actual operator CLI session or controller NETCONF session through which the configuration changes to the Data-Plane are executed.

Physical misconfiguration of wiring between ACP nodes will also not break the ACP: As long as there is a transitive physical path between ACP nodes, the ACP should be able to recover given that it automatically operates across all interfaces of the ACP nodes and automatically determines paths between them.

Attacks against the network via incorrect routing or addressing information for the Data-Plane will not impact the ACP. Even impaired ACP nodes will have a significantly reduced attack surface against malicious misconfiguration because only very limited ACP or interface up/down configuration can affect the ACP, and pending on their specific designs these type of attacks could also be eliminated. See more in Section 9.3 and Section 11.

An ACP node can be a router, switch, controller, NMS host, or any other IPv6 capable node. Initially, it MUST have its ACP certificate, as well as an (empty) ACP Adjacency Table (described in Section 6.3). It then can start to discover ACP neighbors and build the ACP. This is described step by step in the following sections:

### 6.1. Requirements for use of Transport Layer Security (TLS)

The following requirements apply to TLS required or used by ACP components. Applicable ACP components include ACP certificate maintenance via EST, see Section 6.2.5, TLS connections for Certificate Revocation List (CRL) Distribution Point (CRLDP) or Online Certificate Status Protocol (OCSP) responder (if used, see Section 6.2.3) and ACP GRASP (see Section 6.9.2). On ANI nodes these requirements also apply to BRSKI.

TLS MUST comply with [RFC7525] except that TLS 1.2 ([RFC5246]) is REQUIRED and that older versions of TLS MUST NOT be used. TLS 1.3 ([RFC8446]) SHOULD be supported. The choice for TLS 1.2 as the lowest common denominator for the ACP is based on current expected most likely availability across the wide range of candidate ACP node types, potentially with non-agile operating system TCP/IP stacks.

TLS MUST offer TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 and TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 and MUST NOT offer options with less than 256-bit symmetric key strength or hash strength of less than 384 bits. When TLS 1.3 is supported, TLS_AES_256_GCM_SHA384 MUST be offered and TLS_CHACHA20_POLY1305_SHA256 MAY be offered.

TLS MUST also include the "Supported Elliptic Curves" extension, it MUST support the NIST P-256 (secp256r1(22)) and P-384 (secp384r1(24)) curves [RFC8422]. In addition, TLS 1.2 clients SHOULD send an ec_point_format extension with a single element, "uncompressed".

### 6.2. ACP Domain, Certificate and Network

The ACP relies on group security. An ACP domain is a group of nodes that trust each other to participate in ACP operations such as creating ACP secure channels in an autonomous peer-to-peer fashion between ACP domain members via protocols such as IPsec. To authenticate and authorize another ACP member node with access to the ACP Domain, each ACP member requires keying material: An ACP node MUST have a Local Device IDentity (LDevID) certificate, henceforth called the ACP certificate and information about one or more Trust Anchor (TA) as required for the ACP domain membership check (Section 6.2.3).

Manual keying via shared secrets is not usable for an ACP domain because it would require a single shared secret across all current and future ACP domain members to meet the expectation of autonomous, peer-to-peer establishment of ACP secure channels between any ACP domain members. Such a single shared secret would be an inacceptable security weakness. Asymmetric keying material (public keys) without certificates does not provide the mechanisms to authenticate ACP domain membership in an autonomous, peer-to-peer fashion for current and future ACP domain members.

The LDevID certificate is called the ACP certificate. The TA is the Certification Authority (CA) root certificate of the ACP domain.

The ACP does not mandate specific mechanisms by which this keying material is provisioned into the ACP node. It only requires the certificate to comply with Section 6.2.1, specifically to have the acp-node-name as specified in Section 6.2.2 in its domain certificate as well as those of candidate ACP peers. See Appendix A.2 for more information about enrollment or provisioning options.

This document uses the term ACP in many places where the Autonomic Networking reference documents [RFC7575] and [I-D.ietf-anima-reference-model] use the word autonomic. This is done because those reference documents consider (only) fully autonomic networks and nodes, but support of ACP does not require support for other components of autonomic networks except for relying on GRASP and providing security and transport for GRASP. Therefore, the word autonomic might be misleading to operators interested in only the ACP.

[RFC7575] defines the term "Autonomic Domain" as a collection of autonomic nodes. ACP nodes do not need to be fully autonomic, but when they are, then the ACP domain is an autonomic domain. Likewise, [I-D.ietf-anima-reference-model] defines the term "Domain Certificate" as the certificate used in an autonomic domain. The ACP certificate is that domain certificate when ACP nodes are (fully) autonomic nodes. Finally, this document uses the term ACP network to refer to the network created by active ACP nodes in an ACP domain. The ACP network itself can extend beyond ACP nodes through the mechanisms described in Section 8.1.

### 6.2.1. ACP Certificates

ACP certificates MUST be [RFC5280] compliant X.509 v3 ([X.509]) certificates.

ACP nodes MUST support handling ACP certificates, TA certificates and certificate chain certificates (henceforth just called certificates in this section) with RSA public keys and certificates with Elliptic Curve (ECC) public keys.

ACP nodes MUST NOT support certificates with RSA public keys of less than 2048-bit modulus or curves with group order of less than 256-bit. They MUST support certificates with RSA public keys with 2048-bit modulus and MAY support longer RSA keys. They MUST support certificates with ECC public keys using NIST P-256 curves and SHOULD support P-384 and P-521 curves.

ACP nodes MUST NOT support certificates with RSA public keys whose modulus is less than 2048 bits, or certificates whose ECC public keys are in groups whose order is less than 256-bits. RSA signing certificates with 2048-bit public keys MUST be supported, and such certificates with longer public keys MAY be supported. ECDSA certificates using the NIST P-256 curve MUST be supported, and such certificates using the P-384 and P-521 curves SHOULD be supported.

ACP nodes MUST support RSA certificates that are signed by RSA signatures over the SHA-256 digest of the contents, and SHOULD additionally support SHA-384 and SHA-512 digests in such signatures. The same requirements for digest usage in certificate signatures apply to ECDSA certificates, and additionally, ACP nodes MUST support ECDSA signatures on ECDSA certificates.

The ACP certificate SHOULD use an RSA key and an RSA signature when the ACP certificate is intended to be used not only for ACP authentication but also for other purposes. The ACP certificate MAY use an ECC key and an ECDSA signature if the ACP certificate is only used for ACP and ANI authentication and authorization.

Any secure channel protocols used for the ACP as specified in this document or extensions of this document MUST therefore support authentication (e.g. signing) starting with these type of certificates. See [RFC8422] for more information.

The reason for these choices are as follows: As of 2020, RSA is still more widely used than ECC, therefore the MUST for RSA. ECC offers equivalent security at (logarithmically) shorter key lengths (see [RFC8422]). This can be beneficial especially in the presence of constrained bandwidth or constrained nodes in an ACP/ANI network. Some ACP functions such as GRASP peer-2-peer across the ACP require end-to-end/any-to-any authentication/authorization, therefore ECC can only reliably be used in the ACP when it MUST be supported on all ACP nodes. RSA signatures are mandatory to be supported also for ECC certificates because CAs themselves may not support ECC yet.

The ACP certificate SHOULD be used for any authentication between nodes with ACP domain certificates (ACP nodes and NOC nodes) where a required authorization condition is ACP domain membership, such as ACP node to NOC/OAM end-to-end security and ASA to ASA end-to-end security. Section 6.2.3 defines this "ACP domain membership check". The uses of this check that are standardized in this document are for the establishment of hop-by-hop ACP secure channels (Section 6.7) and for ACP GRASP (Section 6.9.2) end-to-end via TLS.

The ACP domain membership check requires a minimum amount of elements in a certificate as described in Section 6.2.3. The identity of a node in the ACP is carried via the acp-node-name as defined in Section 6.2.2.

To support ECDH directly with the key in the ACP certificate, ACP certificates with ECC keys need to indicate to be Elliptic Curve Diffie-Hellman capable (ECDH): If the X.509v3 keyUsage extension is present, the keyAgreement bit must then be set. Note that this option is not required for any of the required ciphersuites in this document and may not be supported by all CA.

Any other fields of the ACP certificate are to be populated as required by [RFC5280]: As long as they are compliant with [RFC5280], any other field of an ACP certificate can be set as desired by the operator of the ACP domain through appropriate ACP registrar/ACP CA procedures. For example, other fields may be required for other purposes that the ACP certificate is intended to be used for (such as elements of a SubjectName).

For further certificate details, ACP certificates may follow the recommendations from [CABFORUM].

For diagnostic and other operational purposes, it is beneficial to copy the device identifying fields of the node's IDevID certificate into the ACP certificate, such as the [X.520], section 6.2.9 "serialNumber" attribute in the subject field distinguished name encoding. Note that this is not the certificate serial number. See also [I-D.ietf-anima-bootstrapping-keyinfra] section 2.3.1. This can be done for example if it would be acceptable for the device’s "serialNumber" to be signaled via the Link Layer Discovery Protocol (LLDP, [LLDP]) because like LLDP signaled information, the ACP certificate information can be retrieved by neighboring nodes without further authentication and be used either for beneficial diagnostics or for malicious attacks. Retrieval of the ACP certificate is possible via a (failing) attempt to set up an ACP secure channel, and the "serialNumber" usually contains device type information that may help to faster determine working exploits/attacks against the device.

Note that there is no intention to constrain authorization within the ACP or autonomic networks using the ACP to just the ACP domain membership check as defined in this document. It can be extended or modified with additional requirements. Such future authorizations can use and require additional elements in certificates or policies or even additional certificates. See the additional check against the id-kp-cmcRA [RFC6402] extended key usage attribute (Section 6.2.5) and for possible future extensions, see Appendix A.9.5.

### 6.2.2. ACP Certificate AcpNodeName

```
acp-node-name = local-part "@" acp-domain-name
local-part = [ acp-address ] [ "+" rsub extensions ]
acp-address = 32HEXDIG | "0" ; HEXDIG as of RFC5234 section B.1
rsub = [ <subdomain> ] ; <subdomain> as of RFC1034, section 3.5
acp-domain-name = ; <domain> ; as of RFC 1034, section 3.5
extensions = *( "+" extension )
extension  = 1*etext  ; future standard definition.
etext      = ALPHA / DIGIT /  ; Printable US-ASCII
             "!" / "#" / "$" / "%" / "&" / "’" /
             "*" / "-" / "/" / "=" / "?" / "^" /
             "_" / "`" / "{" / "|" / "}" / "~"

routing-subdomain = [ rsub "." ] acp-domain-name

Example:
  given an ACP address   of fd89:b714:f3db:0:200:0:6400:0000
  and an ACP domain-name of acp.example.com
  and an rsub extenstion of area51.research

then this results in:
acp-node-name      = fd89b714f3db00000200000064000000
                     +area51.research@acp.example.com
acp-domain-name    = acp.example.com
routing-subdomain  = area51.research.acp.example.com
```

Figure 2: ACP Node Name ABNF

acp-node-name in above Figure 2 is the ABNF ([RFC5234]) definition of the ACP Node Name. An ACP certificate MUST carry this information. It MUST be encoded as a subjectAltName / otherName / AcpNodeName as described in Section 6.2.2.1.

Nodes complying with this specification MUST be able to receive their ACP address through the domain certificate, in which case their own ACP certificate MUST have a 32HEXDIG acp-address field. Acp-address is case insensitive because ABNF HEXDIG is. It is recommended to encode acp-address with lower case letters. Nodes complying with this specification MUST also be able to authenticate nodes as ACP domain members or ACP secure channel peers when they have a 0-value acp-address field and as ACP domain members (but not as ACP secure channel peers) when the acp-address field is omitted from their AcpNodeName. See Section 6.2.3.

acp-domain-name is used to indicate the ACP Domain across which ACP nodes authenticate and authorize each other, for example to build ACP secure channels to each other, see Section 6.2.3. acp-domain-name SHOULD be the FQDN of an Internet domain owned by the network administration of the ACP and ideally reserved to only be used for the ACP. In this specification it serves to be a name for the ACP that ideally is globally unique. When acp-domain-name is a globally unique name, collision of ACP addresses across different ACP domains can only happen due to ULA hash collisions (see Section 6.11.2). Using different acp-domain-names, operators can distinguish multiple ACP even when using the same TA.

To keep the encoding simple, there is no consideration for internationalized acp-domain-names. The acp-node-name is not intended for end user consumption. There is no protection against an operator to pick any domain name for an ACP whether or not the operator can claim to own the domain name. Instead, the domain name only serves as a hash seed for the ULA and for diagnostics to the operator. Therefore, any operator owning only an internationalized domain name should be able to pick an equivalently unique 7-bit ASCII acp-domain-name string representing the internationalized domain name.

"routing-subdomain" is a string that can be constructed from the acp-node-name, and it is used in the hash-creation of the ULA (see below). The presence of the "rsub" component allows a single ACP domain to employ multiple /48 ULA prefixes. See Appendix A.6 for example use-cases.

The optional "extensions" field is used for future standardized extensions to this specification. It MUST be ignored if present and not understood.

The following points explain and justify the encoding choices described:

1. Formatting notes:
   1.1 "rsub" needs to be in the "local-part": If the format just had routing-subdomain as the domain part of the acp-node-name, rsub and acp-domain-name could not be separated from each other to determine in the ACP domain membership check which part is the acp-domain-name and which is solely for creating a different ULA prefix.
   1.2 If both "acp-address" and "rsub" are omitted from AcpNodeName, the "local-part" will have the format "++extension(s)". The two plus characters are necessary so the node can unambiguously parse that both "acp-address" and "rsub" are omitted.
2. The encoding of the ACP domain name and ACP address as described in this section is used for the following reasons:
   2.1 The acp-node-name is the identifier of a node's ACP. It includes the necessary components to identify a node's ACP both from within the ACP as well as from the outside of the ACP.
   2.2 For manual and/or automated diagnostics and backend management of devices and ACPs, it is necessary to have an easily human readable and software parsed standard, single string representation of the information in the acp-node-name. For example, inventory or other backend systems can always identify an entity by one unique string field but not by a combination of multiple fields, which would be necessary if there was no single string representation.
   2.3 If the encoding was not that of such a string, it would be necessary to define a second standard encoding to provide this format (standard string encoding) for operator consumption.
   2.4 Addresses of the form <local>@<domain> have become the preferred format for identifiers of entities in many systems, including the majority of user identification in web or mobile applications such as multi-domain single-sign-on systems.
3. Compatibilities:
   3.1 It should be possible to use the ACP certificate as an LDevID certificate on the system for other uses beside the ACP. Therefore, the information element required for the ACP should be encoded so that it minimizes the possibility of creating incompatibilities with such other uses. The

attributes of the subject field for example are often used in non-ACP applications and should therefore not be occupied by new ACP values.

3.2 The element should not require additional ASN.1 en/decoding, because libraries to access certificate information especially for embedded devices may not support extended ASN.1 decoding beyond predefined, mandatory fields. subjectAltName / otherName is already used with a single string parameter for several otherNames (see [RFC3920], [RFC7585], [RFC4985], [RFC8398]).

3.3 The element required for the ACP should minimize the risk of being misinterpreted by other uses of the LDevID certificate. It also must not be misinterpreted to actually be an email address, hence the use of the otherName / rfc822Name option in the certificate would be inappropriate.

See section 4.2.1.6 of [RFC5280] for details on the subjectAltName field.

#### 6.2.2.1. AcpNodeName ASN.1 Module

The following ASN.1 module normatively specifies the AcpNodeName structure. This specification uses the ASN.1 definitions from [RFC5912] with the 2002 ASN.1 notation used in that document. [RFC5912] updates normative documents using older ASN.1 notation.

```
   ANIMA-ACP-2020
     { iso(1) identified-organization(3) dod(6)
       internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
       id-mod-anima-acpnodename-2020(IANA1) }

   DEFINITIONS IMPLICIT TAGS ::=
   BEGIN

   IMPORTS
     OTHER-NAME
     FROM PKIX1Implicit-2009
       { iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) id-mod(0)
         id-mod-pkix1-implicit-02(59) }

     id-pkix
     FROM PKIX1Explicit-2009
       { iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) id-mod(0)
         id-mod-pkix1-explicit-02(51) } ;

     id-on OBJECT IDENTIFIER ::= { id-pkix 8 }

     AcpNodeNameOtherNames OTHER-NAME ::= { on-AcpNodeName, ... }

     on-AcpNodeName OTHER-NAME ::= {
         AcpNodeName IDENTIFIED BY id-on-AcpNodeName
     }

     id-on-AcpNodeName OBJECT IDENTIFIER ::= { id-on IANA2 }

     AcpNodeName ::= IA5String (SIZE (1..MAX))
      -- AcpNodeName as specified in this document carries the
      -- acp-node-name as specified in the ABNF in Section 6.1.2

   END
```

Figure 3

#### 6.2.3. ACP domain membership check

The following points constitute the ACP domain membership check of a candidate peer via its certificate:

1: The peer has proved ownership of the private key associated with the certificate’s public key. This check is performed by the security association protocol used, for example [RFC7296], section 2.15.

2: The peer's certificate passes certificate path validation as defined in [RFC5280], section 6 against one of the TA associated with the ACP node's ACP certificate (see Section 6.2.4 below). This includes verification of the validity (lifetime) of the certificates in the path.

3: If the peer's certificate indicates a Certificate Revocation List (CRL) Distribution Point (CRLDP) ([RFC5280], section 4.2.1.13) or Online Certificate Status Protocol (OCSP) responder ([RFC5280], section 4.2.2.1), then the peer's certificate MUST be valid according to those mechanisms when they are available: An OCSP check for the peer's certificate across the ACP must succeed or the peer certificate must not be listed in the CRL retrieved from the CRLDP. These mechanisms are not available when the ACP node has no ACP or non-ACP connectivity to retrieve a current CRL or access an OCSP responder and the security association protocol itself has also no way to communicate CRL or OCSP check. Retries to learn revocation via OCSP/CRL SHOULD be made using the same backoff as described in Section 6.7. If and when the ACP node then learns that an ACP peer's certificate is invalid for which rule 3 had to be skipped during ACP secure channel establishment, then the ACP secure channel to that peer MUST be closed even if this peer is the only connectivity to access CRL/OCSP. This applies (of course) to all ACP secure channels to this peer if there are multiple. The ACP secure channel connection MUST be retried periodically to support the case that the neighbor acquires a new, valid certificate.

4: The peer's certificate has a syntactically valid acp-node-name field and the acp-domain-name in that peer's acp-node-name is the same as in this ACP node's certificate (lowercase normalized).

When checking a candidate peer's certificate for the purpose of establishing an ACP secure channel, one additional check is performed:

5: The acp-address field of the candidate peer certificate's AcpNodeName is not omitted but either 32HEXDIG or 0, according to Figure 2.

Technically, ACP secure channels can only be built with nodes that have an acp-address. Rule 5 ensures that this is taken into account during ACP domain membership check.

Nodes with an omitted acp-address field can only use their ACP domain certificate for non-ACP-secure channel authentication purposes. This includes for example NMS type nodes permitted to communicate into the ACP via ACP connect (Section 8.1)

The special value 0 in an ACP certificates acp-address field is used for nodes that can and should determine their ACP address through other mechanisms than learning it through the acp-address field in their ACP certificate. These ACP nodes are permitted to establish ACP secure channels. Mechanisms for those nodes to determine their ACP address are outside the scope of this specification, but this option is defined here so that any ACP nodes can build ACP secure channels to them according to Rule 5.

The optional rsub field of the AcpNodeName is not relevant to the ACP domain membership check because it only serves to structure routing and addressing within an ACP but not to segment mutual authentication/authorization (hence the name "routing subdomain").

In summary:

* Steps 1...4 constitute standard certificate validity verification and private key authentication as defined by [RFC5280] and security association protocols (such as Internet Key Exchange Protocol version 2 IKEv2 [RFC7296] when leveraging certificates.
* Steps 1...4 do not include verification of any pre-existing form of non-public-key-only based identity elements of a certificate such as a web servers domain name prefix often encoded in certificate common name. Step 5 is an equivalent step for the AcpNodeName.
* Step 4 constitutes standard CRL/OCSP checks refined for the case of missing connectivity and limited functionality security association protocols.
* Steps 1...4 authorize to build any secure connection between members of the same ACP domain except for ACP secure channels.
* Step 5 is the additional verification of the presence of an ACP address as necessary for ACP secure channels.
* Steps 1...5 therefore authorize to build an ACP secure channel.

For brevity, the remainder of this document refers to this process only as authentication instead of as authentication and authorization.

[RFC-Editor: Please remove the following paragraph].

Note that the ACP domain membership check does not verify the network layer address of the security association. See [ACPDRAFT], Appendix B.2 for explanations.

#### 6.2.3.1. Realtime clock and Time Validation

An ACP node with a realtime clock in which it has confidence, MUST check the time stamps when performing ACP domain membership check such as the certificate validity period in step 1. and the respective times in step 4 for revocation information (e.g., signingTimes in CMS signatures).

An ACP node without such a realtime clock MAY ignore those time stamp validation steps if it does not know the current time. Such an ACP node SHOULD obtain the current time in a secured fashion, such as via a Network Time Protocol signaled through the ACP. It then ignores time stamp validation only until the current time is known. In the absence of implementing a secured mechanism, such an ACP node MAY use a current time learned in an insecure fashion in the ACP domain membership check.

Current time MAY for example be learned unsecured via NTP ([RFC5905]) over the same link-local IPv6 addresses used for the ACP from neighboring ACP nodes. ACP nodes that do provide NTP insecure over their link-local addresses SHOULD primarily run NTP across the ACP and provide NTP time across the ACP only when they have a trusted time source. Details for such NTP procedures are beyond the scope of this specification.

Beside ACP domain membership check, the ACP itself has no dependency against knowledge of the current time, but protocols and services using the ACP will likely have the need to know the current time. For example, event logging.

### 6.2.4. Trust Anchors (TA)

ACP nodes need TA information according to [RFC5280], section 6.1.1 (d), typically in the form of one or more certificate of the TA to perform certificate path validation as required by Section 6.2.3, rule 2. TA information MUST be provisioned to an ACP node (together with its ACP domain certificate) by an ACP Registrar during initial enrollment of a candidate ACP node. ACP nodes MUST also support renewal of TA information via EST as described below in Section 6.2.5.

The required information about a TA can consist of not only a single, but multiple certificates as required for dealing with CA certificate renewals as explained in Section 4.4 of CMP ([RFC4210]).

A certificate path is a chain of certificates starting at the ACP certificate (leaf/end-entity) followed by zero or more intermediate CA certificates and ending with the TA information, which are typically one or two the self-signed certificates of the TA. The CA that signs the ACP certificate is called the assigning CA. If there are no intermediate CA, then the assigning CA is the TA. Certificate path validation authenticates that the ACP certificate is permitted by a TA associated with the ACP, directly or indirectly via one or more intermediate CA.

Note that different ACP nodes may have different intermediate CA in their certificate path and even different TA. The set of TA for an ACP domain must be consistent across all ACP members so that any ACP node can authenticate any other ACP node. The protocols through which ACP domain membership check rules 1-3 are performed need to support the exchange not only of the ACP nodes certificates, but also exchange of the intermedia TA.

ACP nodes MUST support for the ACP domain membership check the certificate path validation with 0 or 1 intermediate CA. They SHOULD support 2 intermediate CA and two TA (to permit migration to from one TA to another TA).

Certificates for an ACP MUST only be given to nodes that are allowed to be members of that ACP. When the signing CA relies on an ACP Registrar, the CA MUST only sign certificates with acp-node-name through trusted ACP Registrars. In this setup, any existing CA, unaware of the formatting of acp-node-name, can be used.

These requirements can be achieved by using a TA private to the owner of the ACP domain or potentially through appropriate contractual agreements between the involved parties (Registrar and CA). Using public CA is out of scope of this document. [RFC-Editor: please remove the following sentence]. See [ACPDRAFT], Appendix B.3 for further considerations.

A single owner can operate multiple independent ACP domains from the same set of TA. Registrars must then know which ACP a node needs to be enrolled into.

#### 6.2.5. Certificate and Trust Anchor Maintenance

ACP nodes MUST support renewal of their Certificate and TA information via EST and MAY support other mechanisms. See Section 6.1 for TLS requirements. An ACP network MUST have at least one ACP node supporting EST server functionality across the ACP so that EST renewal is useable.

ACP nodes SHOULD be able to remember the IPv6 locator parameters of the O_IPv6_LOCATOR in GRASP of the EST server from which they last renewed their ACP certificate. They SHOULD provide the ability for these EST server parameters to also be set by the ACP Registrar (see Section 6.11.7) that initially enrolled the ACP device with its ACP certificate. When BRSKI (see [I-D.ietf-anima-bootstrapping-keyinfra]) is used, the IPv6 locator of the BRSKI registrar from the BRSKI TLS connection SHOULD be remembered and used for the next renewal via EST if that registrar also announces itself as an EST server via GRASP (see next section) on its ACP address.

The EST server MUST present a certificate that is passing ACP domain membership check in its TLS connection setup (Section 6.2.3, rules 1...4, not rule 5 as this is not for an ACP secure channel setup). The EST server certificate MUST also contain the id-kp-cmcRA [RFC6402] extended key usage attribute and the EST client MUST check its presence.

The additional check against the id-kp-cmcRA extended key usage extension field ensures that clients do not fall prey to an illicit EST server. While such illicit EST servers should not be able to support certificate signing requests (as they are not able to elicit a signing response from a valid CA), such an illicit EST server would be able to provide faked CA certificates to EST clients that need to renew their CA certificates when they expire.

Note that EST servers supporting multiple ACP domains will need to have for each of these ACP domains a separate certificate and respond on a different transport address (IPv6 address and/or TCP port), but this is easily automated on the EST server as long as the CA does not restrict registrars to request certificates with the id-kp-cmcRA extended usage extension for themselves.

#### 6.2.5.1. GRASP objective for EST server

ACP nodes that are EST servers MUST announce their service via GRASP in the ACP through M_FLOOD messages. See [I-D.ietf-anima-grasp], section 2.8.11 for the definition of this message type:

```
Example:

[M_FLOOD, 12340815, h'fd89b714f3db00000200000064000001', 210000,
    [["SRV.est", 4, 255 ],
    [O_IPv6_LOCATOR,
         h'fd89b714f3db00000200000064000001', IPPROTO_TCP, 443]]
]
```

Figure 4: GRASP SRV.est example

The formal definition of the objective in Concise data definition language (CDDL) (see [RFC8610]) is as follows:

```
 flood-message = [M_FLOOD, session-id, initiator, ttl,
                  +[objective, (locator-option / [])]]
                              ; see example above and explanation
                              ; below for initiator and ttl

 objective = ["SRV.est", objective-flags, loop-count,
                                        objective-value]

 objective-flags = sync-only  ; as in GRASP spec
 sync-only       = 4          ; M_FLOOD only requires synchronization
 loop-count      = 255        ; recommended as there is no mechanism
                              ; to discover network diameter.
 objective-value = any        ; reserved for future extensions
```

Figure 5: GRASP SRV.est definition

The objective name "SRV.est" indicates that the objective is an [RFC7030] compliant EST server because "est" is an [RFC6335] registered service name for [RFC7030]. Objective-value MUST be ignored if present. Backward compatible extensions to [RFC7030] MAY be indicated through objective-value. Non [RFC7030] compatible certificate renewal options MUST use a different objective-name. Non-recognized objective-values (or parts thereof if it is a structure partially understood) MUST be ignored.

The M_FLOOD message MUST be sent periodically. The default SHOULD be 60 seconds; the value SHOULD be operator configurable but SHOULD be not smaller than 60 seconds. The frequency of sending MUST be such that the aggregate amount of periodic M_FLOODs from all flooding sources cause only negligible traffic across the ACP. The time-to-live (ttl) parameter SHOULD be 3.5 times the period so that up to three consecutive messages can be dropped before considering an announcement expired. In the example above, the ttl is 210000 msec, 3.5 times 60 seconds. When a service announcer using these parameters unexpectedly dies immediately after sending the M_FLOOD, receivers would consider it expired 210 seconds later. When a receiver tries to connect to this dead service before this timeout, it will experience a failing connection and use that as an indication that the service instance is dead and select another instance of the same service instead (from another GRASP announcement).

The "SRV.est" objective(s) SHOULD only be announced when the ACP node knows that it can successfully communicate with a CA to perform the EST renewal/rekeying operations for the ACP domain.  See also Section 11.

6.2.5.2.  Renewal

When performing renewal, the node SHOULD attempt to connect to the remembered EST server.  If that fails, it SHOULD attempt to connect to an EST server learned via GRASP.  The server with which certificate renewal succeeds SHOULD be remembered for the next renewal.

Remembering the last renewal server and preferring it provides stickiness which can help diagnostics.  It also provides some protection against off-path compromised ACP members announcing bogus information into GRASP.

Renewal of certificates SHOULD start after less than 50% of the domain certificate lifetime so that network operations has ample time to investigate and resolve any problems that causes a node to not renew its domain certificate in time - and to allow prolonged periods of running parts of a network disconnected from any CA.

6.2.5.3.  Certificate Revocation Lists (CRLs)

The ACP node SHOULD support revocation through CRL(s) via HTTP from one or more CRL Distribution Points (CRLDP).  The CRLDP(s) MUST be indicated in the Domain Certificate when used.  If the CRLDP URL uses an IPv6 address (ULA address when using the addressing rules specified in this document), the ACP node will connect to the CRLDP via the ACP.  If the CRLDP uses a domain name, the ACP node will connect to the CRLDP via the Data-Plane.

It is common to use domain names for CRLDP(s), but there is no requirement for the ACP to support DNS.  Any DNS lookup in the Data-Plane is not only a possible security issue, but it would also not indicate whether the resolved address is meant to be reachable across the ACP.  Therefore, the use of an IPv6 address versus the use of a DNS name doubles as an indicator whether or not to reach the CRLDP via the ACP.

A CRLDP can be reachable across the ACP either by running it on a node with ACP or by connecting its node via an ACP connect interface (see Section 8.1).

When using a private PKI for ACP certificates, the CRL may be need-to-know, for example to prohibit insight into the operational practices of the domain by tracking the growth of the CRL. In this case, HTTPS may be chosen to provide confidentiality, especially when making the CRL available via the Data-Plane. Authentication and authorization SHOULD use ACP certificates and ACP domain membership check. The CRLDP MAY omit the CRL verification during authentication of the peer to permit retrieval of the CRL by an ACP node with revoked ACP certificate. This can allow for that (ex) ACP node to quickly discover its ACP certificate revocation. This may violate the desired need-to-know requirement though. ACP nodes MAY support CRLDP operations via HTTPS.

#### 6.2.5.4. Lifetimes

Certificate lifetime may be set to shorter lifetimes than customary (1 year) because certificate renewal is fully automated via ACP and EST. The primary limiting factor for shorter certificate lifetimes is load on the EST server(s) and CA. It is therefore recommended that ACP certificates are managed via a CA chain where the assigning CA has enough performance to manage short lived certificates. See also Section 9.2.4 for discussion about an example setup achieving this. See also [I-D.ietf-acme-star].

When certificate lifetimes are sufficiently short, such as few hours, certificate revocation may not be necessary, allowing to simplify the overall certificate maintenance infrastructure.

See Appendix A.2 for further optimizations of certificate maintenance when BRSKI can be used ("Bootstrapping Remote Secure Key Infrastructures", see [I-D.ietf-anima-bootstrapping-keyinfra]).

#### 6.2.5.5. Re-enrollment

An ACP node may determine that its ACP certificate has expired, for example because the ACP node was powered down or disconnected longer than its certificate lifetime. In this case, the ACP node SHOULD convert to a role of a re-enrolling candidate ACP node.

In this role, the node does maintain the TA and certificate chain associated with its ACP certificate exclusively for the purpose of re-enrollment, and attempts (or waits) to get re-enrolled with a new ACP certificate. The details depend on the mechanisms/protocols used by the ACP Registrars.

Please refer to Section 6.11.7 and [I-D.ietf-anima-bootstrapping-keyinfra] for explanations about ACP Registrars and vouchers as used in the following text. When ACP is intended to be used without BRSKI, the details about BRSKI and vouchers in the following text can be skipped.

When BRSKI is used (i.e.: on ACP nodes that are ANI nodes), the re-enrolling candidate ACP node would attempt to enroll like a candidate ACP node (BRSKI pledge), but instead of using the ACP nodes IDevID certificate, it SHOULD first attempt to use its ACP domain certificate in the BRSKI TLS authentication. The BRSKI registrar MAY honor this certificate beyond its expiration date purely for the purpose of re-enrollment. Using the ACP node's domain certificate allows the BRSKI registrar to learn that node's acp-node-name, so that the BRSKI registrar can re-assign the same ACP address information to the ACP node in the new ACP certificate.

If the BRSKI registrar denies the use of the old ACP certificate, the re-enrolling candidate ACP node MUST re-attempt re-enrollment using its IDevID certificate as defined in BRSKI during the TLS connection setup.

Both when the BRSKI connection is attempted with the old ACP certificate or the IDevID certificate, the re-enrolling candidate ACP node SHOULD authenticate the BRSKI registrar during TLS connection setup based on its existing TA certificate chain information associated with its old ACP certificate. The re-enrolling candidate ACP node SHOULD only fall back to requesting a voucher from the BRSKI registrar when this authentication fails during TLS connection setup. As a countermeasure against attacks that attempt to force the ACP node to forget its prior (expired) certificate and TA, the ACP node should alternate between attempting to re-enroll using its old keying material and attempting to re-enroll with its IDevID and requesting a voucher.

When other mechanisms than BRSKI are used for ACP certificate enrollment, the principles of the re-enrolling candidate ACP node are the same. The re-enrolling candidate ACP node attempts to authenticate any ACP Registrar peers during re-enrollment protocol/ mechanisms via its existing certificate chain/TA information and provides its existing ACP certificate and other identification (such as the IDevID certificate) as necessary to the registrar.

Maintaining existing TA information is especially important when enrollment mechanisms are used that unlike BRSKI do not leverage a mechanism (such as the voucher in BRSKI) to authenticate the ACP registrar and where therefore the injection of certificate failures could otherwise make the ACP node easily attackable remotely by returning the ACP node to a "duckling" state in which it accepts to be enrolled by any network it connects to. The (expired) ACP certificate and ACP TA SHOULD therefore be maintained and attempted to be used as one possible credential for re-enrollment until new keying material is acquired.

When using BRSKI or other protocol/mechanisms supporting vouchers, maintaining existing TA information allows for re-enrollment of expired ACP certificates to be more lightweight, especially in environments where repeated acquisition of vouchers during the lifetime of ACP nodes may be operationally expensive or otherwise undesirable.

#### 6.2.5.6. Failing Certificates

An ACP certificate is called failing in this document, if/when the ACP node to which the certificate was issued can determine that it was revoked (or explicitly not renewed), or in the absence of such explicit local diagnostics, when the ACP node fails to connect to other ACP nodes in the same ACP domain using its ACP certificate. For connection failures to determine the ACP certificate as the culprit, the peer should pass the domain membership check (Section 6.2.3) and other reasons for the connection failure can be excluded because of the connection error diagnostics.

This type of failure can happen during setup/refresh of a secure ACP channel connections or any other use of the ACP certificate, such as for the TLS connection to an EST server for the renewal of the ACP domain certificate.

Example reasons for failing certificates that the ACP node can only discover through connection failure are that the domain certificate or any of its signing certificates could have been revoked or may have expired, but the ACP node cannot self-diagnose this condition directly. Revocation information or clock synchronization may only be available across the ACP, but the ACP node cannot build ACP secure channels because ACP peers reject the ACP node's domain certificate.

ACP nodes SHOULD support the option to determines whether its ACP certificate is failing, and when it does, put itself into the role of a re-enrolling candidate ACP node as explained above (Section 6.2.5.5).

## 6.3. ACP Adjacency Table

To know to which nodes to establish an ACP channel, every ACP node maintains an adjacency table. The adjacency table contains information about adjacent ACP nodes, at a minimum: Node-ID (identifier of the node inside the ACP, see Section 6.11.3 and Section 6.11.5), interface on which neighbor was discovered (by GRASP as explained below), link-local IPv6 address of neighbor on that interface, certificate (including acp-node-name). An ACP node MUST maintain this adjacency table. This table is used to determine to which neighbor an ACP connection is established.

Where the next ACP node is not directly adjacent (i.e., not on a link connected to this node), the information in the adjacency table can be supplemented by configuration. For example, the Node-ID and IP address could be configured. See Section 8.2.

The adjacency table MAY contain information about the validity and trust of the adjacent ACP node's certificate. However, subsequent steps MUST always start with the ACP domain membership check against the peer (see Section 6.2.3).

The adjacency table contains information about adjacent ACP nodes in general, independently of their domain and trust status. The next step determines to which of those ACP nodes an ACP connection should be established.

## 6.4. Neighbor Discovery with DULL GRASP

[RFC-Editor: GRASP draft is in RFC editor queue, waiting for dependencies, including ACP. Please ensure that references to I-D.ietf-anima-grasp that include section number references (throughout this document) will be updated in case any last-minute changes in GRASP would make those section references change.

Discovery Unsolicited Link-Local (DULL) GRASP is a limited subset of GRASP intended to operate across an insecure link-local scope. See section 2.5.2 of [I-D.ietf-anima-grasp] for its formal definition. The ACP uses one instance of DULL GRASP for every L2 interface of the ACP node to discover link level adjacent candidate ACP neighbors. Unless modified by policy as noted earlier (Section 5 bullet point 2.), native interfaces (e.g., physical interfaces on physical nodes) SHOULD be initialized automatically to a state in which ACP discovery can be performed and any native interfaces with ACP neighbors can then be brought into the ACP even if the interface is otherwise not configured. Reception of packets on such otherwise not configured interfaces MUST be limited so that at first only IPv6 StateLess Address Auto Configuration (SLAAC - [RFC4862]) and DULL GRASP work and then only the following ACP secure channel setup packets - but not any other unnecessary traffic (e.g., no other link-local IPv6 transport stack responders for example).

Note that the use of the IPv6 link-local multicast address (ALL_GRASP_NEIGHBORS) implies the need to use Multicast Listener Discovery Version 2 (MLDv2, see [RFC3810]) to announce the desire to receive packets for that address. Otherwise DULL GRASP could fail to operate correctly in the presence of MLD snooping ([RFC4541]) switches that are not ACP supporting/enabled - because those switches would stop forwarding DULL GRASP packets. Switches not supporting MLD snooping simply need to operate as pure L2 bridges for IPv6 multicast packets for DULL GRASP to work.

ACP discovery SHOULD NOT be enabled by default on non-native interfaces. In particular, ACP discovery MUST NOT run inside the ACP across ACP virtual interfaces. See Section 9.3 for further, non-normative suggestions on how to enable/disable ACP at node and interface level. See Section 8.2.2 for more details about tunnels (typical non-native interfaces). See Section 7 for how ACP should be extended on devices operating (also) as L2 bridges.

Note: If an ACP node also implements BRSKI to enroll its ACP certificate (see Appendix A.2 for a summary), then the above considerations also apply to GRASP discovery for BRSKI. Each DULL instance of GRASP set up for ACP is then also used for the discovery of a bootstrap proxy via BRSKI when the node does not have a domain certificate. Discovery of ACP neighbors happens only when the node does have the certificate. The node therefore never needs to discover both a bootstrap proxy and ACP neighbor at the same time.

An ACP node announces itself to potential ACP peers by use of the "AN_ACP" objective. This is a synchronization objective intended to be flooded on a single link using the GRASP Flood Synchronization (M_FLOOD) message. In accordance with the design of the Flood message, a locator consisting of a specific link-local IP address, IP protocol number and port number will be distributed with the flooded objective. An example of the message is informally:

```
[M_FLOOD, 12340815, h'fe80000000000000c0011001feef0000', 210000,
  [["AN_ACP", 4, 1, "IKEv2" ],
   [O_IPv6_LOCATOR,
        h'fe80000000000000c0011001feef0000', IPPROTO_UDP, 15000]]
  [["AN_ACP", 4, 1, "DTLS" ],
   [O_IPv6_LOCATOR,
        h'fe80000000000000c0011001feef0000', IPPROTO_UDP, 17000]]
]
```

Figure 6: GRASP AN_ACP example

The formal CDDL definition is:

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

objective = ["AN_ACP", objective-flags, loop-count,
                                       objective-value]

objective-flags = sync-only ; as in the GRASP specification
sync-only =  4    ; M_FLOOD only requires synchronization
loop-count = 1    ; limit to link-local operation

objective-value = method-name / [ method, *extension ]
method = method-name / [ method-name, *method-param ]
method-name = "IKEv2" / "DTLS" / id
extension = any
method-param = any
id = text .regexp "[A-Za-z@_$]([-.]*[A-Za-z0-9@_$])*"
```

Figure 7: GRASP AN_ACP definition

The objective-flags field is set to indicate synchronization.

The loop-count is fixed at 1 since this is a link-local operation.

In the above example the RECOMMENDED period of sending of the objective is 60 seconds. The indicated ttl of 210000 msec means that the objective would be cached by ACP nodes even when two out of three messages are dropped in transit.

The session-id is a random number used for loop prevention (distinguishing a message from a prior instance of the same message). In DULL this field is irrelevant but has to be set according to the GRASP specification.

The originator MUST be the IPv6 link local address of the originating ACP node on the sending interface.

The method-name in the 'objective-value' parameter is a string indicating the protocol available at the specified or implied locator. It is a protocol supported by the node to negotiate a secure channel. IKEv2 as shown above is the protocol used to negotiate an IPsec secure channel.

Method-params allows to carry method specific parameters. This specification does not define any method-param(s) for "IKEv2" or "DTLS". Method-params for these two methods that are not understood by an ACP node MUST be ignored by it.

extension(s) allows to define method independent parameters. This specification does not define any extensions. Extensions not understood by an ACP node MUST be ignored by it.

The locator-option is optional and only required when the secure channel protocol is not offered at a well-defined port number, or if there is no well-defined port number.

IKEv2 is the actual protocol used to negotiate an Internet Protocol security architecture (IPsec) connection. GRASP therefore indicates "IKEv2" and not "IPsec". If "IPsec" was used, this too could mean use of the obsolete older version IKE (v1) ([RFC2409]). IKEv2 has an IANA assigned port number 500, but in the above example, the candidate ACP neighbor is offering ACP secure channel negotiation via IKEv2 on port 15000 (purely to show through the example that GRASP allows to indicate the port number and it does not have to be the IANA assigned one).

There is no default UDP port for DTLS, it is always locally assigned by the node. For further details about the "DTLS" secure channel protocol, see Section 6.8.4.

If a locator is included, it MUST be an O_IPv6_LOCATOR, and the IPv6 address MUST be the same as the initiator address (these are DULL requirements to minimize third party DoS attacks).

The secure channel methods defined in this document use the objective-values of "IKEv2" and "DTLS". There is no distinction between IKEv2 native and GRE-IKEv2 because this is purely negotiated via IKEv2.

A node that supports more than one secure channel protocol method needs to flood multiple versions of the "AN_ACP" objective so that each method can be accompanied by its own locator-option. This can use a single GRASP M_FLOOD message as shown in Figure 6.

The use of DULL GRASP primarily serves to discover the link-local IPv6 address of candidate ACP peers on subnets. The signaling of the supported secure channel option is primarily for diagnostic purposes, but it is also necessary for discovery when the protocol has no well-known transport address, such as in the case of DTLS. [RFC-Editor: Please remove the following sentence]. See [ACPDRAFT], Appendix B.4.

Note that a node serving both as an ACP node and BRSKI Join Proxy may choose to distribute the "AN_ACP" objective and the respective BRSKI in the same M_FLOOD message, since GRASP allows multiple objectives in one message. This may be impractical though if ACP and BRSKI operations are implemented via separate software modules / ASAs.

The result of the discovery is the IPv6 link-local address of the neighbor as well as its supported secure channel protocols (and non-standard port they are running on). It is stored in the ACP Adjacency Table (see Section 6.3), which then drives the further building of the ACP to that neighbor.

Note that the DULL GRASP objective described intentionally does not include the ACP node's ACP certificate even though this would be useful for diagnostics and to simplify the security exchange in ACP secure channel security association protocols (see Section 6.8). The reason is that DULL GRASP messages are periodically multicasted across IPv6 subnets and full certificates could easily lead to fragmented IPv6 DULL GRASP multicast packets due to the size of a certificate. This would be highly undesirable.

## 6.5. Candidate ACP Neighbor Selection

An ACP node determines to which other ACP nodes in the adjacency table it should attempt to build an ACP connection. This is based on the information in the ACP Adjacency table.

The ACP is established exclusively between nodes in the same domain. This includes all routing subdomains. Appendix A.6 explains how ACP connections across multiple routing subdomains are special.

The result of the candidate ACP neighbor selection process is a list of adjacent or configured autonomic neighbors to which an ACP channel should be established. The next step begins that channel establishment.

## 6.6. Channel Selection

To avoid attacks, initial discovery of candidate ACP peers cannot include any non-protected negotiation. To avoid re-inventing and validating security association mechanisms, the next step after discovering the address of a candidate neighbor can only be to try first to establish a security association with that neighbor using a well-known security association method.

From the use-cases it seems clear that not all type of ACP nodes can or need to connect directly to each other or are able to support or prefer all possible mechanisms. For example, code space limited IoT devices may only support DTLS because that code exists already on them for end-to-end security, but low-end in-ceiling L2 switches may only want to support Media Access Control Security (MacSec, see 802.1AE ([MACSEC]) because that is also supported in their chips. Only a flexible gateway device may need to support both of these mechanisms and potentially more. Note that MacSec is not required by any profiles of the ACP in this specification. Instead, MacSec is mentioned as a likely next interesting secure channel protocol. Note also that the security model allows and requires for any-to-any authentication and authorization between all ACP nodes because there is also end-to-end and not only hop-by-hop authentication for secure channels.

To support extensible secure channel protocol selection without a single common mandatory to implement (MTI) protocol, ACP nodes MUST try all the ACP secure channel protocols it supports and that are feasible because the candidate ACP neighbor also announced them via its AN_ACP GRASP parameters (these are called the "feasible" ACP secure channel protocols).

To ensure that the selection of the secure channel protocols always succeeds in a predictable fashion without blocking, the following rules apply:

* An ACP node may choose to attempt to initiate the different feasible ACP secure channel protocols it supports according to its local policies sequentially or in parallel, but it MUST support acting as a responder to all of them in parallel.
* Once the first ACP secure channel protocol connection to a specific peer IPv6 address passes peer authentication, the two peers know each other's certificate because those ACP certificates are used by all secure channel protocols for mutual authentication. The peer with the higher Node-ID in the AcpNodeName of its ACP certificate takes on the role of the Decider towards the peer. The other peer takes on the role of the Follower. The Decider selects which secure channel protocol to ultimately use.
* The Follower becomes passive: it does not attempt to further initiate ACP secure channel protocol connections with the Decider and does not consider it to be an error when the Decider closes secure channels. The Decider becomes the active party, continues to attempt setting up secure channel protocols with the Follower. This process terminates when the Decider arrives at the "best" ACP secure channel connection option that also works with the Follower ("best" from the Deciders point of view).
* A peer with a "0" acp-address in its AcpNodeName takes on the role of Follower when peering with a node that has a non-"0" acp-address (note that this specification does not fully define the

behavior of ACP secure channel negotiation for nodes with a "0" ACP address field, it only defines interoperability with such ACP nodes).

In a simple example, ACP peer Node 1 attempts to initiate an IPsec via IKEv2 connection to peer Node 2. The IKEv2 authentication succeeds. Node 1 has the lower ACP address and becomes the Follower. Node 2 becomes the Decider. IKEv2 might not be the preferred ACP secure channel protocol for the Decider Node 2. Node 2 would therefore proceed to attempt secure channel setups with (in its view) more preferred protocol options (e.g., DTLS/UDP). If any such preferred ACP secure channel connection of the Decider succeeds, it would close the IPsec connection. If Node 2 has no preferred protocol option over IPsec, or no such connection attempt from Node 2 to Node 1 succeeds, Node 2 would keep the IPsec connection and use it.

The Decider SHOULD NOT send actual payload packets across a secure channel until it has decided to use it. The Follower MAY delay linking the ACP secure channel into the ACP virtual interface until it sees the first payload packet from the Decider up to a maximum of 5 seconds to avoid unnecessarily linking a secure channel that will be terminated as undesired by the Decider shortly afterwards.

The following sequence of steps show this example in more detail. Each step is tagged with [<step#>{:<connection>}]. The connection is included to more easily distinguish which of the two competing connections the step belongs to, one initiated by Node 1, one initiated by Node 2.

```
[1]    Node 1 sends GRASP AN_ACP message to announce itself

[2]    Node 2 sends GRASP AN_ACP message to announce itself

[3]    Node 2 receives [1] from Node 1

[4:C1] Because of [3], Node 2 starts as initiator on its
       preferred secure channel protocol towards Node 1.
       Connection C1.

[5]    Node 1 receives [2] from Node 2

[6:C2] Because of [5], Node 1 starts as initiator on its
       preferred secure channel protocol towards Node 2.
       Connection C2.

[7:C1] Node1 and Node2 have authenticated each others
       certificate on connection C1 as valid ACP peers.

[8:C1] Node 1 certificate has lower ACP Node-ID than Node2, therefore
       Node 1 considers itself the Follower and Node 2 the Decider
       on connection C1. Connection setup C1 is completed.

[9]    Node 1 refrains from attempting any further secure channel
       connections to Node 2 (the Decider) as learned from [2]
       because it knows from [8:C1] that it is the Follower
       relative to Node 1.

[10:C2] Node1 and Node2 have authenticated each others
       certificate on connection C2 (like [7:C1]).

[11:C2] Node 1 certificate has lower ACP Node-ID than Node2,
        therefore Node 1 considers itself the Follower and Node 2 the
        Decider on connection C2, but they also identify that C2 is
        to the same mutual peer as their C1, so this has no further
        impact: the roles Decider and Follower where already assigned
        between these two peers by [8:C1].

[12:C2] Node 2 (the Decider) closes C1. Node 1 is fine with this,
        because of its role as the Follower (from [8:C1]).

[13]    Node 2 (the Decider) and Node 1 (the Follower) start data
        transfer across C2, which makes it become a secure channel
        for the ACP.
```

Figure 8: Secure Channel sequence of steps

All this negotiation is in the context of an "L2 interface". The Decider and Follower will build ACP connections to each other on every "L2 interface" that they both connect to. An autonomic node MUST NOT assume that neighbors with the same L2 or link-local IPv6 addresses on different L2 interfaces are the same node. This can only be determined after examining the certificate after a successful security association attempt.

The Decider SHOULD NOT suppress attempting a particular ACP secure channel protocol connection on one L2 interface because this type of ACP secure channel connection has failed to the peer with the same ACP certificate on another L2 interface: Not only the supported ACP secure channel protocol options may be different on the same ACP peer across different L2 interfaces, but also error conditions may cause inconsistent failures across different L2 interfaces. Avoiding such connection attempt optimizations can therefore help to increase robustness in the case of errors.

## 6.7. Candidate ACP Neighbor verification

Independent of the security association protocol chosen, candidate ACP neighbors need to be authenticated based on their domain certificate. This implies that any secure channel protocol MUST support certificate based authentication that can support the ACP domain membership check as defined in Section 6.2.3. If it fails, the connection attempt is aborted and an error logged. Attempts to reconnect MUST be throttled. The RECOMMENDED default is exponential base 2 backoff with an initial retransmission time (IRT) of 10 seconds and a maximum retransmission time (MRT) of 640 seconds.

Failure to authenticate an ACP neighbor when acting in the role of a responder of the security authentication protocol MUST NOT impact the attempts of the ACP node to attempt establishing a connection as an initiator. Only failed connection attempts as an initiator must cause throttling. This rule is meant to increase resilience of secure channel creation. Section 6.6 shows how simultaneous mutual secure channel setup collisions are resolved.

## 6.8. Security Association (Secure Channel) protocols

This section describes how ACP nodes establish secured data connections to automatically discovered or configured peers in the ACP. Section 6.4 above described how IPv6 subnet adjacent peers are discovered automatically. Section 8.2 describes how non IPv6 subnet adjacent peers can be configured.

Section 6.13.5.2 describes how secure channels are mapped to virtual IPv6 subnet interfaces in the ACP. The simple case is to map every ACP secure channel into a separate ACP point-to-point virtual interface Section 6.13.5.2.1. When a single subnet has multiple ACP peers this results in multiple ACP point-to-point virtual interfaces across that underlying multi-party IPv6 subnet. This can be optimized with ACP multi-access virtual interfaces (Section 6.13.5.2.2) but the benefits of that optimization may not justify the complexity of that option.

### 6.8.1. General considerations

Due to Channel Selection (Section 6.6), ACP can support an evolving set of security association protocols and does not require support for a single network wide MTI. ACP nodes only need to implement those protocols required to interoperate with their candidate peers, not with potentially any node in the ACP domain. See Section 6.8.5 for an example of this.

The degree of security required on every hop of an ACP network needs to be consistent across the network so that there is no designated "weakest link" because it is that "weakest link" that would otherwise become the designated point of attack. When the secure channel protection on one link is compromised, it can be used to send/receive packets across the whole ACP network. Therefore, even though the security association protocols can be different, their minimum degree of security should be comparable.

Secure channel protocols do not need to always support arbitrary L3 connectivity between peers, but can leverage the fact that the standard use case for ACP secure channels is an L2 adjacency. Hence, L2 dependent mechanisms could be adopted for use as secure channel association protocols:

L2 mechanisms such as strong encrypted radio technologies or [MACSEC] may offer equivalent encryption and the ACP security association protocol may only be required to authenticate ACP domain membership of a peer and/or derive a key for the L2 mechanism. Mechanisms to auto-discover and associate ACP peers leveraging such underlying L2 security are possible and desirable to avoid duplication of encryption, but none are specified in this document.

Strong physical security of a link may stand in where cryptographic security is infeasible. As there is no secure mechanism to automatically discover strong physical security solely between two peers, it can only be used with explicit configuration and that configuration too could become an attack vector. This document therefore only specifies with ACP connect (Section 8.1) one explicitly configured mechanism without any secure channel association protocol - for the case where both the link and the nodes attached to it have strong physical security.

#### 6.8.2. Common requirements

The authentication of peers in any security association protocol MUST use the ACP certificate according to Section 6.2.3. Because auto-discovery of candidate ACP neighbors via GRASP (see Section 6.4) as specified in this document does not communicate the neighbors ACP certificate, and ACP nodes may not (yet) have any other network connectivity to retrieve certificates, any security association protocol MUST use a mechanism to communicate the certificate directly instead of relying on a referential mechanism such as communicating only a hash and/or URL for the certificate.

A security association protocol MUST use Forward Secrecy (whether inherently or as part of a profile of the security association protocol).

Because the ACP payload of legacy protocol payloads inside the ACP and hop-by-hop ACP flooded GRASP information is unencrypted, the ACP secure channel protocol requires confidentiality. Symmetric encryption for the transmission of secure channel data MUST use encryption schemes considered to be security wise equal to or better than 256-bit key strength, such as AES256. There MUST NOT be support for NULL encryption.

Security association protocols typically only signal the End Entity certificate (e.g. the ACP certificate) and any possible intermediate CA certificates for successful mutual authentication. The TA has to be mutually known and trusted and therefore its certificate does not need to be signaled for successful mutual authentication. Nevertheless, for use with ACP secure channel setup, there SHOULD be the option to include the TA certificate in the signaling to aid troubleshooting, see Section 9.1.1.

Signaling of TA certificates may not be appropriate when the deployment is relying on a security model where the TA certificate content is considered confidential and only its hash is appropriate for signaling. ACP nodes SHOULD have a mechanism to select whether the TA certificate is signaled or not. Assuming that both options are possible with a specific secure channel protocol.

An ACP secure channel MUST immediately be terminated when the lifetime of any certificate in the chain used to authenticate the neighbor expires or becomes revoked. This may not be standard behavior in secure channel protocols because the certificate authentication may only influence the setup of the secure channel in these protocols, but may not be re-validated during the lifetime of the secure connection in the absence of this requirement.

When specifying an additional security association protocol for ACP secure channels beyond those covered in this document, protocol options SHOULD be eliminated that are not necessary to support devices that are expected to be able to support the ACP to minimize implementation complexity. For example, definitions for security protocols often include old/inferior security options required only to interoperate with existing devices that will not be able to update to the currently preferred security options. Such old/inferior security options do not need to be supported when a security association protocol is first specified for the ACP, strengthening the "weakest link" and simplifying ACP implementation overhead.

#### 6.8.3. ACP via IPsec

An ACP node announces its ability to support IPsec, negotiated via IKEv2, as the ACP secure channel protocol using the "IKEv2" objective-value in the "AN_ACP" GRASP objective.

The ACP usage of IPsec and IKEv2 mandates a profile with a narrow set of options of the current standards-track usage guidance for IPsec [RFC8221] and IKEv2 [RFC8247]. These option result in stringent security properties and can exclude deprecated/legacy algorithms because there is no need for interoperability with legacy equipment for ACP secure channels. Any such backward compatibility would lead only to increased attack surface and implementation complexity, for no benefit.

##### 6.8.3.1. Native IPsec

An ACP node that is supporting native IPsec MUST use IPsec in tunnel mode, negotiated via IKEv2, and with IPv6 payload (e.g., ESP Next Header of 41). It MUST use local and peer link-local IPv6 addresses for encapsulation. Manual keying MUST NOT be used, see Section 6.2. Traffic Selectors are:

```
TSi = (0, 0-65535, :: - FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF)

TSr = (0, 0-65535, :: - FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF)
```

IPsec tunnel mode is required because the ACP will route/forward packets received from any other ACP node across the ACP secure channels, and not only its own generated ACP packets. With IPsec transport mode (and no additional encapsulation header in the ESP payload), it would only be possible to send packets originated by the ACP node itself because the IPv6 addresses of the ESP must be the same as that of the outer IPv6 header.

6.8.3.1.1. RFC8221 (IPsec/ESP)

ACP IPsec implementations MUST comply with [RFC8221] (and its updates). The requirements from above and this section amend and superseded its requirements.

The IP Authentication Header (AH) MUST NOT be used (because it does not provide confidentiality).

For the required ESP encryption algorithms in section 5 of [RFC8221] the following guidance applies:

* ENCR_NULL AH MUST NOT be used (because it does not provide confidentiality).
* ENCR_AES_GCM_16 is the only MTI ESP encryption algorithm for ACP via IPsec/ESP (it is already listed as MUST in [RFC8221]).
* ENCR_AES_CBC with AUTH_HMAC_SHA2_256_128 (as the ESP authentication algorithm) and ENCR_AES_CCM_8 MAY be supported. If either provides higher performance than ENCR_AES_GCM_16 it SHOULD be supported.
* ENCR_CHACHA20_POLY1305 SHOULD be supported at equal or higher performance than ENCR_AES_GCM_16. If that performance is not feasible, it MAY be supported.

IKEv2 indicates an order for the offered algorithms. The algorithms SHOULD be ordered by performance. The first algorithm supported by both sides is generally chosen.

Explanations:

* There is no requirement to interoperate with legacy equipment in ACP secure channels, so a single MTI encryption algorithm for IPsec in ACP secure channels is sufficient for interoperability and allows for the most lightweight implementations.
* ENCR_AES_GCM_16 is an authenticated encryption with associated data (AEAD) cipher mode, so no additional ESP authentication algorithm is needed, simplifying the MTI requirements of IPsec for the ACP.

* There is no MTI requirement for the support of ENCR_AES_CBC because ENCR_AES_GCM_16 is assumed to be feasible with less cost/ higher performance in modern devices hardware accelerated implementations compared to ENCR-AES_CBC.
* ENCR_CHACHA20_POLY1305 is mandatory in [RFC8221] because of its target use as a fallback algorithm in case weaknesses in AES are uncovered. Unfortunately, there is currently no way to automatically propagate across an ACP a policy to disallow use of AES based algorithms, so this target benefit of ENCR_CHACHA20_POLY1305 cannot fully be adopted yet for the ACP. Therefore, this algorithm is only recommended. Changing from AES to this algorithm at potentially big drop in performance could also render the ACP inoperable. Therefore, the performance requirement against this algorithm so that it could become an effective security backup to AES for the ACP once a policy to switch over to it or prefer it is available in an ACP framework.

[RFC8221] allows for 128-bit or 256-bit AES keys. This document mandates that only 256-bit AES keys MUST be supported.

When [RFC8221] is updated, ACP implementations will need to consider legacy interoperability, and the IPsec WG has generally done a very good job of taking that into account in its recommendations.

###### 6.8.3.1.2. RFC8247 (IKEv2)

[RFC8247] provides a baseline recommendation for mandatory to implement ciphers, integrity checks, pseudo-random-functions and Diffie-Hellman mechanisms. Those recommendations, and the recommendations of subsequent documents apply well to the ACP. Because IKEv2 for ACP secure channels is sufficient to be implemented in control plane software, rather than in ASIC hardware, and ACP nodes supporting IKEv2 are not assumed to be code-space constrained, and because existing IKEv2 implementations are expected to support [RFC8247] recommendations, this documents makes no attempt to simplify its recommendations for use with the ACP.

See [IKEV2IANA] for IANA IKEv2 parameter names used in this text.

ACP Nodes supporting IKEv2 MUST comply with [RFC8247] amended by the following requirements which constitute a policy statement as permitted by [RFC8247].

To signal the ACP certificate chain (including TA) as required by Section 6.8.2, "X.509 Certificate - Signature" payload in IKEv2 can be used. It is mandatory according to [RFC7296] section 3.6.

ACP nodes SHOULD set up IKEv2 to only use the ACP certificate and TA when acting as an IKEv2 responder on the IPv6 link local address and port number indicated in the AN_ACP DULL GRASP announcements (see Section 6.4).

When CERTREQ is received from a peer, and does not indicate any of this ACP nodes TA certificates, the ACP node SHOULD ignore the CERTREQ and continue sending its certificate chain including its TA as subject to the requirements and explanations in Section 6.8.2. This will not result in successful mutual authentication but assists diagnostics.

Note that with IKEv2, failing authentication will only result in the responder receiving the certificate chain from the initiator, but not vice versa. Because ACP secure channel setup is symmetric (see Section 6.7), every non-malicious ACP neighbor will attempt to connect as an initiator though, allowing to obtain the diagnostic information about the neighbors certificate.

In IKEv2, ACP nodes are identified by their ACP address. The ID_IPv6_ADDR IKEv2 identification payload MUST be used and MUST convey the ACP address. If the peer's ACP certificate includes a 32HEXDIG ACP address in the acp-node-name (not "0" or omitted), the address in the IKEv2 identification payload MUST match it. See Section 6.2.3 for more information about "0" or omitted ACP address fields in the acp-node-name.

IKEv2 authentication MUST use authentication method 14 ("Digital Signature") for ACP certificates; this authentication method can be used with both RSA and ECDSA certificates, indicated by an ASN.1 object AlgorithmIdentifier.

The Digital Signature hash SHA2-512 MUST be supported (in addition to SHA2-256).

The IKEv2 Diffie-Hellman key exchange group 19 (256-bit random ECP), MUST be supported. Reason: ECC provides a similar security level to finite-field (MODP) key exchange with a shorter key length, so is generally preferred absent other considerations.

#### 6.8.3.2. IPsec with GRE encapsulation

In network devices it is often more common to implement high performance virtual interfaces on top of GRE encapsulation than on top of a "native" IPsec association (without any other encapsulation than those defined by IPsec). On those devices it may be beneficial to run the ACP secure channel on top of GRE protected by the IPsec association.

The requirements for ESP/IPsec/IKEv2 with GRE are the same as for native IPsec (see Section 6.8.3.1) except that IPsec transport mode and next protocol GRE (47) are to be negotiated. Tunnel mode is not required because of GRE. Traffic Selectors are:

TSi = (47, 0-65535, Initiator-IPv6-LL-addr ... Initiator-IPv6-LL-addr)

TSr = (47, 0-65535, Responder-IPv6-LL-addr ... Responder-IPv6-LL-addr)

If IKEv2 initiator and responder support IPsec over GRE, it will be preferred over native IPsec because of the way how IKEv2 negotiates transport mode (as used by this IPsec over GRE profile) versus tunnel mode as used by native IPsec (see [RFC7296], section 1.3.1). The ACP IPv6 traffic has to be carried across GRE according to [RFC7676].

### 6.8.4. ACP via DTLS

This document defines the use of ACP via DTLS, on the assumption that it is likely the first transport encryption supported in some classes of constrained devices: DTLS is commonly used in constrained devices when IPsec is not. Code-space on those devices may be also be too limited to support more than the minimum number of required protocols.

An ACP node announces its ability to support DTLS version 1.2 ([RFC6347]) compliant with the requirements defined in this document as an ACP secure channel protocol in GRASP through the "DTLS" objective-value in the "AN_ACP" objective (see Section 6.4).

To run ACP via UDP and DTLS, a locally assigned UDP port is used that is announced as a parameter in the GRASP AN_ACP objective to candidate neighbors. This port can also be any newer version of DTLS as long as that version can negotiate a DTLS v1.2 connection in the presence of an DTLS v1.2 only peer.

All ACP nodes supporting DTLS as a secure channel protocol MUST adhere to the DTLS implementation recommendations and security considerations of BCP 195, BCP 195 [RFC7525] except with respect to the DTLS version. ACP nodes supporting DTLS MUST support DTLS 1.2. They MUST NOT support older versions of DTLS.

Unlike for IPsec, no attempts are made to simplify the requirements of the BCP 195 recommendations because the expectation is that DTLS would be using software-only implementations where the ability to reuse of widely adopted implementations is more important than minimizing the complexity of a hardware accelerated implementation which is known to be important for IPsec.

DTLS v1.3 ([I-D.ietf-tls-dtls13]) is "backward compatible" with DTLS v1.2 (see section 1. of DTLS v1.3). A DTLS implementation supporting both DTLS v1.2 and DTLS v1.3 does comply with the above requirements of negotiating to DTLS v1.2 in the presence of a DTLS v1.2 only peer, but using DTLS v1.3 when booth peers support it.

Version v1.2 is the MTI version of DTLS in this specification because

* There is more experience with DTLS v1.2 across the spectrum of target ACP nodes.
* Firmware of lower end, embedded ACP nodes may not support a newer version for a long time.
* There are significant changes of DTLS v1.3, such as a different record layer requiring time to gain implementation and deployment experience especially on lower end, code space limited devices.
* The existing BCP [RFC7525] for DTLS v1.2 may equally take longer time to be updated with experience from a newer DTLS version.
* There are no significant use-case relevant benefits of DTLS v1.3 over DTLS v1.2 in the context of the ACP options for DTLS. For example, signaling performance improvements for session setup in DTLS v1.3 is not important for the ACP given the long-lived nature of ACP secure channel connections and the fact that DTLS connections are mostly link-local (short RTT).

Nevertheless, newer versions of DTLS, such as DTLS v1.3 have stricter security requirements and use of the latest standard protocol version is for IETF security standards in general recommended. Therefore, ACP implementations are advised to support all the newer versions of DTLS that can still negotiate down to DTLS v1.2.

[RFC-editor: if by the time of AUTH48, DTLS 1.3 would have evolved to be an RFC, then not only would the references to the DTLS v1.3 draft be changed to the RFC number, but that RFC is then going to be put into the normative list of references and the above paragraph is going to be amended to say: Implementations SHOULD support [DTLSv1.3-RFC]. This is not done right now, because there is no benefit in potentially waiting in RFC-editor queue for that RFC given how the text already lays out a non-normative desire to support DTLSv1.3.]

There is no additional session setup or other security association besides this simple DTLS setup. As soon as the DTLS session is functional, the ACP peers will exchange ACP IPv6 packets as the payload of the DTLS transport connection. oAny DTLS defined security association mechanisms such as re-keying are used as they would be for any transport application relying solely on DTLS.

#### 6.8.5. ACP Secure Channel Profiles

As explained in the beginning of Section 6.6, there is no single secure channel mechanism mandated for all ACP nodes. Instead, this section defines two ACP profiles (baseline and constrained) for ACP nodes that do introduce such requirements.

An ACP node supporting the "baseline" profile MUST support IPsec natively and MAY support IPsec via GRE. An ACP node supporting the "constrained" profile node that cannot support IPsec MUST support DTLS. An ACP node connecting an area of constrained ACP nodes with an area of baseline ACP nodes needs to support IPsec and DTLS and supports therefore the baseline and constrained profile.

Explanation: Not all type of ACP nodes can or need to connect directly to each other or are able to support or prefer all possible secure channel mechanisms. For example, code space limited IoT devices may only support DTLS because that code exists already on them for end-to-end security, but high-end core routers may not want to support DTLS because they can perform IPsec in accelerated hardware but would need to support DTLS in an underpowered CPU forwarding path shared with critical control plane operations. This is not a deployment issue for a single ACP across these type of nodes as long as there are also appropriate gateway ACP nodes that support sufficiently many secure channel mechanisms to allow interconnecting areas of ACP nodes with a more constrained set of secure channel protocols. On the edge between IoT areas and high-end core networks, general-purpose routers that act as those gateways and that can support a variety of secure channel protocols is the norm already.

IPsec natively with tunnel mode provides the shortest encapsulation overhead. GRE may be preferred by legacy implementations because the virtual interfaces required by ACP design in conjunction with secure channels have in the past more often been implemented for GRE than purely for native IPsec.

ACP nodes need to specify in documentation the set of secure ACP mechanisms they support and should declare which profile they support according to above requirements.

### 6.9. GRASP in the ACP

#### 6.9.1. GRASP as a core service of the ACP

The ACP MUST run an instance of GRASP inside of it. It is a key part of the ACP services. The function in GRASP that makes it fundamental as a service of the ACP is the ability to provide ACP wide service discovery (using objectives in GRASP).

ACP provides IP unicast routing via the RPL routing protocol (see Section 6.12).

The ACP does not use IP multicast routing nor does it provide generic IP multicast services (the handling of GRASP link-local multicast messages is explained in Section 6.9.2). Instead, the ACP provides service discovery via the objective discovery/announcement and negotiation mechanisms of the ACP GRASP instance (services are a form of objectives). These mechanisms use hop-by-hop reliable flooding of GRASP messages for both service discovery (GRASP M_DISCOVERY messages) and service announcement (GRASP M_FLOOD messages).

See Appendix A.5 for discussion about this design choice of the ACP.

#### 6.9.2. ACP as the Security and Transport substrate for GRASP

In the terminology of GRASP ([I-D.ietf-anima-grasp]), the ACP is the security and transport substrate for the GRASP instance run inside the ACP ("ACP GRASP").

This means that the ACP is responsible for ensuring that this instance of GRASP is only sending messages across the ACP GRASP virtual interfaces. Whenever the ACP adds or deletes such an interface because of new ACP secure channels or loss thereof, the ACP needs to indicate this to the ACP instance of GRASP. The ACP exists also in the absence of any active ACP neighbors. It is created when the node has a domain certificate, and continues to exist even if all of its neighbors cease operation.

In this case ASAs using GRASP running on the same node would still need to be able to discover each other's objectives. When the ACP does not exist, ASAs leveraging the ACP instance of GRASP via APIs MUST still be able to operate, and MUST be able to understand that there is no ACP and that therefore the ACP instance of GRASP cannot operate.

The following explanation how ACP acts as the security and transport substrate for GRASP is visualized in Figure 9 below.

GRASP unicast messages inside the ACP always use the ACP address. Link-local addresses from the ACP VRF MUST NOT be used inside objectives. GRASP unicast messages inside the ACP are transported via TLS. See Section 6.1 for TLS requirements. TLS mutual authentication MUST use the ACP domain membership check defined in (Section 6.2.3).

GRASP link-local multicast messages are targeted for a specific ACP virtual interface (as defined Section 6.13.5) but are sent by the ACP into an ACP GRASP virtual interface that is constructed from the TCP connection(s) to the IPv6 link-local neighbor address(es) on the underlying ACP virtual interface. If the ACP GRASP virtual interface has two or more neighbors, the GRASP link-local multicast messages are replicated to all neighbor TCP connections.

TCP and TLS connections for GRASP in the ACP use the IANA assigned TCP port for GRASP (7107). Effectively the transport stack is expected to be TLS for connections from/to the ACP address (e.g., global scope address(es)) and TCP for connections from/to link-local addresses on the ACP virtual interfaces. The latter ones are only used for flooding of GRASP messages.

```
       ..............................ACP..............................
       .                                                             .
       .         /-GRASP-flooding-\         ACP GRASP instance       .
       .        /                  \                                 A
       .    GRASP      GRASP      GRASP                              C
       .  link-local   unicast  link-local                           P
       .   multicast  messages   multicast                           .
       .   messages      |       messages                            .
       .      |          |          |                                .
       ...............................................................
       .      v          v          v    ACP security and transport  .
       .      |          |          |    substrate for GRASP         .
       .      |          |          |                                .
       .      |       ACP GRASP     |       - ACP GRASP              A
       .      |       Loopback      |         Loopback interface     C
       .      |       interface     |       - ACP-cert auth          P
       .      |         TLS         |                                .
       .   ACP GRASP     |       ACP GRASP  - ACP GRASP virtual      .
       .   subnet1       |       subnet2      virtual interfaces     .
       .     TCP         |         TCP                               .
       .      |          |          |                                .
       ...............................................................
       .      |          |          |   ^^^ Users of ACP (GRASP/ASA) .
       .      |          |          |   ACP interfaces/addressing    .
       .      |          |          |                                .
       .      |          |          |                                A
       .      | ACP-Loopback Interf.|      <- ACP Loopback interface C
       .      |      ACP-address    |       - address (global ULA)   P
       .    subnet1      |        subnet2  <- ACP virtual interfaces .
       .  link-local     |      link-local  - link-local addresses   .
       ...............................................................
       .      |          |          |   ACP VRF                      .
       .      |     RPL-routing     | virtual routing and forwarding .
       .      |   /IP-Forwarding\   |                                A
       .      |  /               \  |                                C
       .  ACP IPv6 packets   ACP IPv6 packets                        P
       .      |/                   \|                                .
       .    IPsec/DTLS        IPsec/DTLS  - ACP-cert auth            .
       ...............................................................
                |                   |   Data-Plane
                |                   |
                |                   |     - ACP secure channel
            link-local        link-local  - encapsulation addresses
              subnet1            subnet2  - Data-Plane interfaces
                |                   |
             ACP-Nbr1            ACP-Nbr2
```

Figure 9: ACP as security and transport substrate for GRASP

#### 6.9.2.1. Discussion

TCP encapsulation for GRASP M_DISCOVERY and M_FLOOD link local messages is used because these messages are flooded across potentially many hops to all ACP nodes and a single link with even temporary packet loss issues (e.g., WiFi/Powerline link) can reduce the probability for loss free transmission so much that applications would want to increase the frequency with which they send these messages.  Such shorter periodic retransmission of datagrams would result in more traffic and processing overhead in the ACP than the hop-by-hop reliable retransmission mechanism by TCP and duplicate elimination by GRASP.

TLS is mandated for GRASP non-link-local unicast because the ACP secure channel mandatory authentication and encryption protects only against attacks from the outside but not against attacks from the inside: Compromised ACP members that have (not yet) been detected and removed (e.g., via domain certificate revocation / expiry).

If GRASP peer connections were to use just TCP, compromised ACP members could simply eavesdrop passively on GRASP peer connections for whom they are on-path ("man in the middle" - MITM) or intercept and modify them.  With TLS, it is not possible to completely eliminate problems with compromised ACP members, but attacks are a lot more complex:

Eavesdropping/spoofing by a compromised ACP node is still possible because in the model of the ACP and GRASP, the provider and consumer of an objective have initially no unique information (such as an identity) about the other side which would allow them to distinguish a benevolent from a compromised peer.  The compromised ACP node would simply announce the objective as well, potentially filter the original objective in GRASP when it is a MITM and act as an application level proxy.  This of course requires that the compromised ACP node understand the semantics of the GRASP negotiation to an extent that allows it to proxy it without being detected, but in an ACP environment this is quite likely public knowledge or even standardized.

The GRASP TLS connections are run the same as any other ACP traffic through the ACP secure channels.  This leads to double authentication/encryption, which has the following benefits:

* Secure channel methods such as IPsec may provide protection against additional attacks, for example reset-attacks.
* The secure channel method may leverage hardware acceleration and there may be little or no gain in eliminating it.

* There is no different security model for ACP GRASP from other ACP traffic. Instead, there is just another layer of protection against certain attacks from the inside which is important due to the role of GRASP in the ACP.

### 6.10. Context Separation

The ACP is in a separate context from the normal Data-Plane of the node. This context includes the ACP channels' IPv6 forwarding and routing as well as any required higher layer ACP functions.

In classical network system, a dedicated VRF is one logical implementation option for the ACP. If possible by the systems software architecture, separation options that minimize shared components are preferred, such as a logical container or virtual machine instance. The context for the ACP needs to be established automatically during bootstrap of a node. As much as possible it should be protected from being modified unintentionally by ("Data-Plane") configuration.

Context separation improves security, because the ACP is not reachable from the Data-Plane routing or forwarding table(s). Also, configuration errors from the Data-Plane setup do not affect the ACP.

### 6.11. Addressing inside the ACP

The channels explained above typically only establish communication between two adjacent nodes. In order for communication to happen across multiple hops, the autonomic control plane requires ACP network wide valid addresses and routing. Each ACP node creates a Loopback interface with an ACP network wide unique address (prefix) inside the ACP context (as explained in in Section 6.10). This address may be used also in other virtual contexts.

With the algorithm introduced here, all ACP nodes in the same routing subdomain have the same /48 ULA prefix. Conversely, ULA global IDs from different domains are unlikely to clash, such that two ACP networks can be merged, as long as the policy allows that merge. See also Section 10.1 for a discussion on merging domains.

Links inside the ACP only use link-local IPv6 addressing, such that each node's ACP only requires one routable address prefix.

#### 6.11.1. Fundamental Concepts of Autonomic Addressing

* Usage: Autonomic addresses are exclusively used for self-management functions inside a trusted domain. They are not used for user traffic. Communications with entities outside the

trusted domain use another address space, for example normally managed routable address space (called "Data-Plane" in this document).

* Separation: Autonomic address space is used separately from user address space and other address realms. This supports the robustness requirement.
* Loopback-only: Only ACP Loopback interfaces (and potentially those configured for "ACP connect", see Section 8.1) carry routable address(es); all other interfaces (called ACP virtual interfaces) only use IPv6 link local addresses. The usage of IPv6 link local addressing is discussed in [RFC7404].
* Use-ULA: For Loopback interfaces of ACP nodes, we use ULA with L=1 (as defined in section 3.1 of [RFC4193]). Note that the random hash for ACP Loopback addresses uses the definition in Section 6.11.2 and not the one of [RFC4193] section 3.2.2.
* No external connectivity: They do not provide access to the Internet. If a node requires further reaching connectivity, it should use another, traditionally managed address scheme in parallel.
* Addresses in the ACP are permanent, and do not support temporary addresses as defined in [RFC4941].
* Addresses in the ACP are not considered sensitive on privacy grounds because ACP nodes are not expected to be end-user hosts and ACP addresses do therefore not represent end-users or groups of end-users. All ACP nodes are in one (potentially federated) administrative domain. They are assumed to be to be candidate hosts of ACP traffic amongst each other or transit thereof. There are no transit nodes less privileged to know about the identity of other hosts in the ACP. Therefore, ACP addresses do not need to be pseudo-random as discussed in [RFC7721]. Because they are not propagated to untrusted (non ACP) nodes and stay within a domain (of trust), we also consider them not to be subject to scanning attacks.

The ACP is based exclusively on IPv6 addressing, for a variety of reasons:

* Simplicity, reliability and scale: If other network layer protocols were supported, each would have to have its own set of security associations, routing table and process, etc.
* Autonomic functions do not require IPv4: Autonomic functions and autonomic service agents are new concepts. They can be exclusively built on IPv6 from day one. There is no need for backward compatibility.
* OAM protocols do not require IPv4: The ACP may carry OAM protocols. All relevant protocols (SNMP, TFTP, SSH, SCP, RADIUS, Diameter, NETCONF ...) are available in IPv6. See also [RFC8368] for how ACP could be made to interoperate with IPv4 only OAM.

Further explanation about the addressing and routing related reasons for the choice of the autonomous ACP addressing can be found in Section 6.13.5.1.

### 6.11.2. The ACP Addressing Base Scheme

The Base ULA addressing scheme for ACP nodes has the following format:

```
  8      40                       2                     78
+--+-------------------------+------+------------------------------+
|fd| hash(routing-subdomain) | Type |      (sub-scheme)            |
+--+-------------------------+------+------------------------------+
```

Figure 10: ACP Addressing Base Scheme

The first 48-bits follow the ULA scheme, as defined in [RFC4193], to which a type field is added:

* "fd" identifies a locally defined ULA address.
* The 40-bits ULA "global ID" (term from [RFC4193]) for ACP addresses carried in the acp-node-name in the ACP certificates are the first 40-bits of the SHA256 hash of the routing subdomain from the same acp-node-name. In the example of Section 6.2.2, the routing subdomain is "area51.research.acp.example.com" and the 40-bits ULA "global ID" 89b714f3db.
* When creating a new routing-subdomain for an existing autonomic network, it MUST be ensured, that rsub is selected so the resulting hash of the routing-subdomain does not collide with the hash of any pre-existing routing-subdomains of the autonomic network. This ensures that ACP addresses created by registrars for different routing subdomains do not collide with each other.
* To allow for extensibility, the fact that the ULA "global ID" is a hash of the routing subdomain SHOULD NOT be assumed by any ACP node during normal operations. The hash function is only executed during the creation of the certificate. If BRSKI is used, then the BRSKI registrar will create the acp-node-name in response to the EST Certificate Signing Request (CSR) Attribute Request message by the pledge.

* Establishing connectivity between different ACP (different acp-domain-name) is outside the scope of this specification. If it is being done through future extensions, then the rsub of all routing-subdomains across those autonomic networks need to be selected so the resulting routing-subdomain hashes do not collide. For example, a large cooperation with its own private TA may want to create different autonomic networks that initially should not be able to connect but where the option to do so should be kept open. When taking this future possibility into account, it is easy to always select rsub so that no collisions happen.
* Type: This field allows different address sub-schemes. This addresses the "upgradability" requirement. Assignment of types for this field will be maintained by IANA.

The sub-scheme may imply a range or set of addresses assigned to the node, this is called the ACP address range/set and explained in each sub-scheme.

Please refer to Section 6.11.7 and Appendix A.1 for further explanations why the following Sub-Addressing schemes are used and why multiple are necessary.

The following summarizes the addressing Sub-Schemes:

| Type | Name | F-bit | Z | V-bits | Prefix |
|---|---|---|---|---|---|
| 0x00 | ACP-Zone | N/A | 0 | 1 bit | /127 |
| 0x00 | ACP-Manual | N/A | 1 | N/A | /64 |
| 0x01 | ACP-VLong-8 | 0 | N/A | 8 bits | /120 |
| 0x01 | ACP-VLong-16 | 1 | N/A | 16 bits | /112 |
| 0x10 | Reserved / For future definition/allocation | | | | |
| 0x11 | Reserved / For future definition/allocation | | | | |

Figure 11: Addressing Sub-Schemes

F-Bit and Z are two encoding fields explained below for the Sub-Schemes that introduce/use them. V-bits is the number of bits of addresses allocated to the ACP node. Prefix is the prefix the ACP node is announcing into the RPL routing protocol.

### 6.11.3. ACP Zone Addressing Sub-Scheme (ACP-Zone)

This sub-scheme is used when the Type field of the base scheme is 0x00 and the Z bit is 0x0.

```
                64                                     64
+-----------------+---+---------++-----------------------------+---+
|  (base scheme)  | Z | Zone-ID ||           Node-ID             |
|                 |   |         || Registrar-ID |   Node-Number| V |
+-----------------+---+---------++--------------+--------------+---+
         50         1     13             48            15         1
```

Figure 12: ACP Zone Addressing Sub-Scheme

The fields are defined as follows:

* Type: MUST be 0x0.
* Z: MUST be 0x0.
* Zone-ID: A value for a network zone.
* Node-ID: A unique value for each node.

The 64-bit Node-ID must be unique across the ACP domain for each node. It is derived and composed as follows:

* Registrar-ID (48-bit): A number unique inside the domain that identifies the ACP registrar which assigned the Node-ID to the node. One or more domain-wide unique identifiers of the ACP registrar can be used for this purpose. See Section 6.11.7.2.
* Node-Number: Number to make the Node-ID unique. This can be sequentially assigned by the ACP Registrar owning the Registrar-ID.
* V (1-bit): Virtualization bit: 0: Indicates the ACP itself ("ACP node base system); 1: Indicates the optional "host" context on the ACP node (see below).

In the ACP Zone Addressing Sub-Scheme, the ACP address in the certificate has V field as all zero bits.

The ACP address set of the node includes addresses with any Zone-ID value and any V value. No two nodes in the same ACP can have the same Node-ID, but different Zone-IDs.

The Virtual bit in this sub-scheme allows the easy addition of the ACP as a component to existing systems without causing problems in the port number space between the services in the ACP and the existing system. V:0 is the ACP router (autonomic node base system), V:1 is the host with pre-existing transport endpoints on it that could collide with the transport endpoints used by the ACP router. The ACP host could for example have a p2p virtual interface with the V:0 address as its router into the ACP. Depending on the software design of ASAs, which is outside the scope of this specification, they may use the V:0 or V:1 address.

The location of the V bit(s) at the end of the address allows the announcement of a single prefix for each ACP node. For example, in a network with 20,000 ACP nodes, this avoid 20,000 additional routes in the routing table.

It is RECOMMENDED that only Zone-ID 0 is used unless it is meant to be used in conjunction with operational practices for partial/ incremental adoption of the ACP as described in Section 9.4.

Note: Zones and Zone-ID as defined here are not related to [RFC4007] zones or zone_id. ACP zone addresses are not scoped (reachable only from within an RFC4007 zone) but reachable across the whole ACP. An RFC4007 zone_id is a zone index that has only local significance on a node, whereas an ACP Zone-ID is an identifier for an ACP zone that is unique across that ACP.

### 6.11.4. ACP Manual Addressing Sub-Scheme (ACP-Manual)

This sub-scheme is used when the Type field of the base scheme is 0x00 and the Z bit is 0x1.

```
                64                             64
+---------------------+---+----------++-----------------------------+
|    (base scheme)    | Z | Subnet-ID||     Interface Identifier    |
+---------------------+---+----------++-----------------------------+
         50           1    13
```

Figure 13: ACP Manual Addressing Sub-Scheme

The fields are defined as follows:

* Type: MUST be 0x0.
* Z: MUST be 0x1.
* Subnet-ID: Configured subnet identifier.
* Interface Identifier.

This sub-scheme is meant for "manual" allocation to subnets where the other addressing schemes cannot be used. The primary use case is for assignment to ACP connect subnets (see Section 8.1.1).

"Manual" means that allocations of the Subnet-ID need to be done today with pre-existing, non-autonomic mechanisms. Every subnet that uses this addressing sub-scheme needs to use a unique Subnet-ID (unless some anycast setup is done).

The Z bit field was added to distinguish Zone addressing and manual addressing sub-schemes without requiring one more bit in the base scheme and therefore allowing for the Vlong scheme (described below) to have one more bit available.

Manual addressing sub-scheme addresses SHOULD NOT be used in ACP certificates. Any node capable to build ACP secure channels and permitted by Registrar policy to participate in building ACP secure channels SHOULD receive an ACP address (prefix) from one of the other ACP addressing sub-schemes. Nodes not capable (or permitted) to participate in ACP secure channels can connect to the ACP via ACP connect interfaces of ACP edge nodes (see Section 8.1), without setting up an ACP secure channel. Their ACP certificate MUST omit the acp-address field to indicate that their ACP certificate is only usable for non- ACP secure channel authentication, such as end-to-end transport connections across the ACP or Data-Plane.

Address management of ACP connect subnets is done using traditional assignment methods and existing IPv6 protocols. See Section 8.1.3 for details. Therefore, the notion of V-bit many addresses assigned to the ACP nodes does not apply to this Sub-Scheme.

### 6.11.5. ACP Vlong Addressing Sub-Scheme (ACP-VLong-8/ACP-VLong-16

This sub-scheme is used when the Type field of the base scheme is 0x01.

```
          50                               78
+---------------------++-----------------------------+----------+
|    (base scheme)    ||               Node-ID                  |
|                     || Registrar-ID |F| Node-Number|         V |
+---------------------++--------------+--------------+----------+
          50                  46        1   23/15          8/16
```

Figure 14: ACP Vlong Addressing Sub-Scheme

This addressing scheme foregoes the Zone-ID field to allow for larger, flatter routed networks (e.g., as in IoT) with 8421376 Node-Numbers (2^23+2^15). It also allows for up to 2^16 (i.e. 65536) different virtualized addresses within a node, which could be used to address individual software components in an ACP node.

The fields are the same as in the Zone-ID sub-scheme with the following refinements:

* F: format bit. This bit determines the format of the subsequent bits.
* V: Virtualization bit: this is a field that is either 8 or 16 bits. For F=0, it is 8 bits, for F=1 it is 16 bits. The V bits are assigned by the ACP node. In the ACP certificate's ACP address Section 6.2.2, the V-bits are always set to 0.
* Registrar-ID: To maximize Node-Number and V, the Registrar-ID is reduced to 46-bits. One or more domain-wide unique identifiers of the ACP registrar can be used for this purpose. See Section 6.11.7.2.
* The Node-Number is unique to each ACP node. There are two formats for the Node-Number. When F=0, the node-number is 23 bits, for F=1 it is 15 bits. Each format of node-number is considered to be in a unique number space.

The F=0 bit format addresses are intended to be used for "general purpose" ACP nodes that would potentially have a limited number (< 256) of clients (ASA/Autonomic Functions or legacy services) of the ACP that require separate V(irtual) addresses.

The F=1 bit Node-Numbers are intended for ACP nodes that are ACP edge nodes (see Section 8.1.1) or that have a large number of clients requiring separate V(irtual) addresses. For example, large SDN controllers with container modular software architecture (see Section 8.1.2).

In the Vlong addressing sub-scheme, the ACP address in the certificate has all V field bits as zero. The ACP address set for the node includes any V value.

### 6.11.6. Other ACP Addressing Sub-Schemes

Before further addressing sub-schemes are defined, experience with the schemes defined here should be collected. The schemes defined in this document have been devised to allow hopefully sufficiently flexible setup of ACPs for a variety of situation. These reasons also lead to the fairly liberal use of address space: The Zone Addressing Sub-Scheme is intended to enable optimized routing in large networks by reserving bits for Zone-ID's. The Vlong addressing sub-scheme enables the allocation of 8/16-bit of addresses inside individual ACP nodes. Both address spaces allow distributed, uncoordinated allocation of node addresses by reserving bits for the registrar-ID field in the address.

#### 6.11.7. ACP Registrars

ACP registrars are responsible to enroll candidate ACP nodes with ACP certificates and associated trust anchor(s). They are also responsible that an acp-node-name field is included in the ACP certificate carrying the ACP domain name and the ACP nodes ACP address prefix. This address prefix is intended to persist unchanged through the lifetime of the ACP node.

Because of the ACP addressing sub-schemes, an ACP domain can have multiple distributed ACP registrars that do not need to coordinate for address assignment. ACP registrars can also be sub-CAs, in which case they can also assign ACP certificates without dependencies against a (shared) TA (except during renewals of their own certificates).

ACP registrars are PKI registration authorities (RA) enhanced with the handling of the ACP certificate specific fields. They request certificates for ACP nodes from a Certification Authority through any appropriate mechanism (out of scope in this document, but required to be BRSKI for ANI registrars). Only nodes that are trusted to be compliant with the requirements against registrar described in this section can be given the necessary credentials to perform this RA function, such as credentials for the BRSKI connection to the CA for ANI registrars.

##### 6.11.7.1. Use of BRSKI or other Mechanism/Protocols

Any protocols or mechanisms may be used by ACP registrars, as long as the resulting ACP certificate and TA certificate(s) allow to perform the ACP domain membership described in Section 6.2.3 with other ACP domain members, and meet the ACP addressing requirements for its acp-node-name as described further below in this section.

An ACP registrar could be a person deciding whether to enroll a candidate ACP node and then orchestrating the enrollment of the ACP certificate and associated TA, using command line or web based commands on the candidate ACP node and TA to generate and sign the ACP certificate and configure certificate and TA onto the node.

The only currently defined protocol for ACP registrars is BRSKI ([I-D.ietf-anima-bootstrapping-keyinfra]). When BRSKI is used, the ACP nodes are called ANI nodes, and the ACP registrars are called BRSKI or ANI registrars. The BRSKI specification does not define the handling of the acp-node-name field because the rules do not depend on BRSKI but apply equally to any protocols/mechanisms an ACP registrar may use.

#### 6.11.7.2. Unique Address/Prefix allocation

ACP registrars MUST NOT allocate ACP address prefixes to ACP nodes via the acp-node-name that would collide with the ACP address prefixes of other ACP nodes in the same ACP domain. This includes both prefixes allocated by the same ACP registrar to different ACP nodes as well as prefixes allocated by other ACP registrars for the same ACP domain.

To support such unique address allocation, an ACP registrar MUST have one or more 46-bit identifiers unique across the ACP domain which is called the Registrar-ID. Allocation of Registrar-ID(s) to an ACP registrar can happen through OAM mechanisms in conjunction with some database / allocation orchestration.

ACP registrars running on physical devices with known globally unique EUI-48 MAC address(es) can use the lower 46 bits of those address(es) as unique Registrar-IDs without requiring any external signaling/configuration (the upper two bits, V and U are not uniquely assigned but functional). This approach is attractive for distributed, non-centrally administered, lightweight ACP registrar implementations. There is no mechanism to deduce from a MAC address itself whether it is actually uniquely assigned. Implementations need to consult additional offline information before making this assumption. For example by knowing that a particular physical product/MIC-chip is guaranteed to use globally unique assigned EUI-48 MAC address(es).

When the candidate ACP device (called Pledge in BRSKI) is to be enrolled into an ACP domain, the ACP registrar needs to allocate a unique ACP address to the node and ensure that the ACP certificate gets a acp-node-name field (Section 6.2.2) with the appropriate information - ACP domain-name, ACP-address, and so on. If the ACP registrar uses BRSKI, it signals the ACP acp-node-name field to the Pledge via the EST /csrattrs command (see [I-D.ietf-anima-bootstrapping-keyinfra], section 5.9.2 - "EST CSR Attributes").

[RFC-Editor: please update reference to section 5.9.2 accordingly with latest BRSKI draft at time of publishing, or RFC]

#### 6.11.7.3. Addressing Sub-Scheme Policies

The ACP registrar selects for the candidate ACP node a unique address prefix from an appropriate ACP addressing sub-scheme, either a zone addressing sub-scheme prefix (see Section 6.11.3), or a Vlong addressing sub-scheme prefix (see Section 6.11.5). The assigned ACP address prefix encoded in the acp-node-name field of the ACP certificate indicates to the ACP node its ACP address information.

The sub-addressing scheme indicates the prefix length: /127 for zone address sub-scheme, /120 or /112 for Vlong address sub-scheme. The first address of the prefix is the ACP address. All other addresses in the prefix are for other uses by the ACP node as described in the zone and Vlong addressing sub scheme sections. The ACP address prefix itself is then signaled by the ACP node into the ACP routing protocol (see Section 6.12) to establish IPv6 reachability across the ACP.

The choice of addressing sub-scheme and prefix-length in the Vlong address sub-scheme is subject to ACP registrar policy. It could be an ACP domain wide policy, or a per ACP node or per ACP node type policy. For example, in BRSKI, the ACP registrar is aware of the IDevID certificate of the candidate ACP node, which typically contains a "serialNumber" attribute in the subject field distinguished name encoding that is often indicating the node's vendor and device type and can be used to drive a policy selecting an appropriate addressing sub-scheme for the (class of) node(s).

ACP registrars SHOULD default to allocate ACP zone sub-address scheme addresses with Zone-ID 0.

ACP registrars that are aware of the IDevID certificate of a candidate ACP device SHOULD be able to choose the zone vs. Vlong sub-address scheme for ACP nodes based on the [X.520] "serialNumber" attribute in the subject field distinguished name encoding of the IDevID certificate, for example by the PID (Product Identifier) part which identifies the product type, or the complete "serialNumber". The PID for example could identify nodes that allow for specialized ASA requiring multiple addresses or non-autonomic VMs for services and those nodes could receive Vlong sub-address scheme ACP addresses.

In a simple allocation scheme, an ACP registrar remembers persistently across reboots its currently used Registrar-ID and for each addressing scheme (Zone with Zone-ID 0, Vlong with /112, Vlong with /120), the next Node-Number available for allocation and increases it during successful enrollment to an ACP node. In this simple allocation scheme, the ACP registrar would not recycle ACP address prefixes from no longer used ACP nodes.

If allocated addresses cannot be remembered by registrars, then it is necessary to either use a new value for the Register-ID field in the ACP addresses, or determine allocated ACP addresses from determining the addresses of reachable ACP nodes, which is not necessarily the set of all ACP nodes. Non-tracked ACP addresses can be reclaimed by revoking or not renewing their certificates and instead handing out new certificate with new addresses (for example with a new Registrar-ID value). Note that such strategies may require coordination amongst registrars.

#### 6.11.7.4. Address/Prefix Persistence

When an ACP certificate is renewed or rekeyed via EST or other mechanisms, the ACP address/prefix in the acp-node-name field MUST be maintained unless security issues or violations of the unique address assignment requirements exist or are suspected by the ACP registrar.

ACP address information SHOULD be maintained even when the renewing/rekeying ACP registrar is not the same as the one that enrolled the prior ACP certificate. See Section 9.2.4 for an example.

ACP address information SHOULD also be maintained even after an ACP certificate did expire or failed. See Section 6.2.5.5 and Section 6.2.5.6.

#### 6.11.7.5. Further Details

Section 9.2 discusses further informative details of ACP registrars: What interactions registrars need, what parameters they require, certificate renewal and limitations, use of sub-CAs on registrars and centralized policy control.

## 6.12. Routing in the ACP

Once ULA address are set up all autonomic entities should run a routing protocol within the autonomic control plane context. This routing protocol distributes the ULA created in the previous section for reachability. The use of the autonomic control plane specific context eliminates the probable clash with Data-Plane routing tables and also secures the ACP from interference from the configuration mismatch or incorrect routing updates.

The establishment of the routing plane and its parameters are automatic and strictly within the confines of the autonomic control plane. Therefore, no explicit configuration is required.

All routing updates are automatically secured in transit as the channels of the ACP are encrypted, and this routing runs only inside the ACP.

The routing protocol inside the ACP is RPL ([RFC6550]). See Appendix A.4 for more details on the choice of RPL.

RPL adjacencies are set up across all ACP channels in the same domain including all its routing subdomains. See Appendix A.6 for more details.

#### 6.12.1. ACP RPL Profile

The following is a description of the RPL profile that ACP nodes need to support by default. The format of this section is derived from [I-D.ietf-roll-applicability-template].

##### 6.12.1.1. Overview

RPL Packet Information (RPI) defined in [RFC6550], section 11.2 defines the data packet artefacts required or beneficial in forwarding of packets routed by RPL. This profile does not use RPI for better compatibility with accelerated hardware forwarding planes which most often does not support the Hop-by-Hop headers used for RPI, but also to avoid the overhead of the RPI header on the wire and cost of adding/removing them.

###### 6.12.1.1.1. Single Instance

To avoid the need for RPI, the ACP RPL profile uses a simple destination prefix based routing/forwarding table. To achieve this, the profiles uses only one RPL instanceID. This single instanceID can contain only one Destination Oriented Directed Acyclic Graph (DODAG), and the routing/forwarding table can therefore only calculate a single class of service ("best effort towards the primary NOC/root") and cannot create optimized routing paths to accomplish latency or energy goals between any two nodes.

This choice is a compromise. Consider a network that has multiple NOCs in different locations. Only one NOC will become the DODAG root. Traffic to and from other NOCs has to be sent through the DODAG (shortest path tree) rooted in the primary NOC. Depending on topology, this can be an annoyance from a latency point of view or from minimizing network path resources, but this is deemed to be acceptable given how ACP traffic is "only" network management/control traffic. See Appendix A.9.4 for more details.

Using a single instanceID/DODAG does not introduce a single point of failure, as the DODAG will reconfigure itself when it detects Data-Plane forwarding failures including choosing a different root when the primary one fails.

The benefit of this profile, especially compared to other IGPs is that it does not calculate routes for node reachable through the same interface as the DODAG root. This RPL profile can therefore scale to much larger number of ACP nodes in the same amount of compute and memory than other routing protocols. Especially on nodes that are leafs of the topology or those close to those leafs.

#### 6.12.1.1.2. Reconvergence

In RPL profiles where RPL Packet Information (RPI, see Section 6.12.1.13) is present, it is also used to trigger reconvergence when misrouted, for example looping, packets are recognized because of their RPI data. This helps to minimize RPL signaling traffic especially in networks without stable topology and slow links.

The ACP RPL profile instead relies on quick reconverging the DODAG by recognizing link state change (down/up) and triggering reconvergence signaling as described in Section 6.12.1.7. Since links in the ACP are assumed to be mostly reliable (or have link layer protection against loss) and because there is no stretch according to Section 6.12.1.7, loops caused by loss of RPL routing protocol signaling packets should be exceedingly rare.

In addition, there are a variety of mechanisms possible in RPL to further avoid temporary loops RECOMMENDED to be used for the ACPL RPL profile: DODAG Information Objects (DIOs) SHOULD be sent 2 or 3 times to inform children when losing the last parent. The technique in [RFC6550] section 8.2.2.6. (Detaching) SHOULD be favored over that in section 8.2.2.5., (Poisoning) because it allows local connectivity. Nodes SHOULD select more than one parent, at least 3 if possible, and send Destination Advertisement Objects (DAO)s to all of them in parallel.

Additionally, failed ACP tunnels can be quickly discovered trough the secure channel protocol mechanisms such as IKEv2 Dead Peer Detection. This can function as a replacement for a Low-power and Lossy Networks' (LLN's) Expected Transmission Count (ETX) feature that is not used in this profile. A failure of an ACP tunnel should immediately signal the RPL control plane to pick a different parent.

#### 6.12.1.2. RPL Instances

Single RPL instance. Default RPLInstanceID = 0.

#### 6.12.1.3. Storing vs. Non-Storing Mode

RPL Mode of Operations (MOP): MUST support mode 2 - "Storing Mode of Operations with no multicast support". Implementations MAY support mode 3 ("... with multicast support" as that is a superset of mode 2). Note: Root indicates mode in DIO flow.

#### 6.12.1.4. DAO Policy

Proactive, aggressive DAO state maintenance:

* Use K-flag in unsolicited DAO indicating change from previous information (to require DAO-ACK).
* Retry such DAO DAO-RETRIES(3) times with DAO- ACK_TIME_OUT(256ms) in between.

#### 6.12.1.5. Path Metric

Use Hopcount according to [RFC6551]. Note that this is solely for diagnostic purposes as it is not used by the objective function.

#### 6.12.1.6. Objective Function

Objective Function (OF): Use OF0 [RFC6552]. No use of metric containers.

rank_factor: Derived from link speed: <= 100Mbps: LOW_SPEED_FACTOR(5), else HIGH_SPEED_FACTOR(1)

This is a simple rank differentiation between typical "low speed" or "IoT" links that commonly max out at 100 Mbps and typical infrastructure links with speeds of 1 Gbps or higher. Given how the path selection for the ACP focusses only on reachability but not on path cost optimization, no attempts at finer grained path optimization are made.

#### 6.12.1.7. DODAG Repair

Global Repair: we assume stable links and ranks (metrics), so there is no need to periodically rebuild the DODAG. The DODAG version is only incremented under catastrophic events (e.g., administrative action).

Local Repair: As soon as link breakage is detected, the ACP node send No-Path DAO for all the targets that were reachable only via this link. As soon as link repair is detected, the ACP node validates if this link provides a better parent. If so, a new rank is computed by the ACP node and it sends new DIO that advertise the new rank. Then it sends a DAO with a new path sequence about itself.

When using ACP multi-access virtual interfaces, local repair can be triggered directly by peer breakage, see Section 6.13.5.2.2.

stretch_rank: none provided ("not stretched").

Data Path Validation: Not used.

Trickle: Not used.

#### 6.12.1.8. Multicast

Not used yet but possible because of the selected mode of operations.

#### 6.12.1.9. Security

[RFC6550] security not used, substituted by ACP security.

Because the ACP links already include provisions for confidentiality and integrity protection, their usage at the RPL layer would be redundant, and so RPL security is not used.

#### 6.12.1.10. P2P communications

Not used.

#### 6.12.1.11. IPv6 address configuration

Every ACP node (RPL node) announces an IPv6 prefix covering the addresses assigned to the ACP node via the AcpNodeName. The prefix length depends on the addressing sub-scheme of the acp-address, /127 for Zone Addressing Sub-Scheme and /112 or /120 for Vlong addressing sub-scheme. See Section 6.11 for more details.

Every ACP node MUST install a black hole (aka null) route if there are unused parts of the ACP address space assigned to the ACP node via its AcpNodeName. This is superseded by longer prefixes assigned to interfaces for the address space actually used by the node. For example, when the node has an ACP-VLong-8 address space, it installs a /120 black hole route. If it then for example only uses the ACP address (first address from the space), it would assign that address via a /128 address prefix to the ACP loopback interface (see Section 6.13.5.1). None of those longer prefixes are announced into RPL.

For ACP-Manual address prefixes configured on an ACP node, for example for ACP connect subnets (see Section 8.1.1), the node announces the /64 subnet prefix.

#### 6.12.1.12. Administrative parameters

Administrative Preference ([RFC6550], 3.2.6 - to become root): Indicated in DODAGPreference field of DIO message.

* Explicit configured "root": 0b100
* ACP registrar (Default): 0b011
* ACP-connect (non-registrar): 0b010
* Default: 0b001.

#### 6.12.1.13. RPL Packet Information

RPI is not required in the ACP RPL profile for the following reasons.

One RPI option is the RPL Source Routing Header (SRH) [RFC6554] which is not necessary because the ACP RPL profile uses storing mode where each hop has the necessary next-hop forwarding information.

The simpler RPL Option header [RFC6553] is also not necessary in this profile, because it uses a single RPL instance and data path validation is also not used.

#### 6.12.1.14. Unknown Destinations

Because RPL minimizes the size of the routing and forwarding table, prefixes reachable through the same interface as the RPL root are not known on every ACP node. Therefore, traffic to unknown destination addresses can only be discovered at the RPL root. The RPL root SHOULD have attach safe mechanisms to operationally discover and log such packets.

As this requirement places additional constraints on the Data-Plane functionality of the RPL root, it does not apply to "normal" nodes that are not configured to have special functionality (i.e., the administrative parameter from Section 6.12.1.12 has value 0b001). If the ACP network is degraded to the point where there are no nodes that could be configured as root, registrar, or ACP-connect nodes, it is possible that the RPL root (and thus the ACP as a whole) would be unable to detect traffic to unknown destinations. However, in the absence of nodes with administrative preference other than 0b001, there is also unlikely to be a way to get diagnostic information out of the ACP, so detection of traffic to unknown destinations would not be actionable anyway.

### 6.13. General ACP Considerations

Since channels are by default established between adjacent neighbors, the resulting overlay network does hop-by-hop encryption. Each node decrypts incoming traffic from the ACP, and encrypts outgoing traffic to its neighbors in the ACP. Routing is discussed in Section 6.12.

#### 6.13.1. Performance

There are no performance requirements against ACP implementations defined in this document because the performance requirements depend on the intended use case. It is expected that full autonomic node with a wide range of ASA can require high forwarding plane performance in the ACP, for example for telemetry. Implementations of ACP to solely support traditional/SDN style use cases can benefit from ACP at lower performance, especially if the ACP is used only for critical operations, e.g., when the Data-Plane is not available. The design of the ACP as specified in this document is intended to support a wide range of performance options: It is intended to allow software-only implementations at potentially low performance, but can also support high performance options. See [RFC8368] for more details.

#### 6.13.2. Addressing of Secure Channels

In order to be independent of the Data-Plane routing and addressing, the GRASP discovered ACP secure channels use IPv6 link local addresses between adjacent neighbors. Note: Section 8.2 specifies extensions in which secure channels are configured tunnels operating over the Data-Plane, so those secure channels cannot be independent of the Data-Plane.

To avoid that Data-Plane configuration can impact the operations of the IPv6 (link-local) interface/address used for ACP channels, appropriate implementation considerations are required. If the IPv6 interface/link-local address is shared with the Data-Plane, it needs to be impossible to unconfigure/disable it through configuration. Instead of sharing the IPv6 interface/link-local address, a separate (virtual) interface with a separate IPv6 link-local address can be used. For example, the ACP interface could be run over a separate MAC address of an underlying L2 (Ethernet) interface. For more details and options, see Appendix A.9.2.

Note that other (non-ideal) implementation choices may introduce additional undesired dependencies against the Data-Plane. For example, shared code and configuration of the secure channel protocols (IPsec / DTLS).

### 6.13.3. MTU

The MTU for ACP secure channels MUST be derived locally from the underlying link MTU minus the secure channel encapsulation overhead.

ACP secure Channel protocols do not need to perform MTU discovery because they are built across L2 adjacencies - the MTU on both sides connecting to the L2 connection are assumed to be consistent. Extensions to ACP where the ACP is for example tunneled need to consider how to guarantee MTU consistency. This is an issue of tunnels, not an issue of running the ACP across a tunnel. Transport stacks running across ACP can perform normal PMTUD (Path MTU Discovery). Because the ACP is meant to prioritize reliability over performance, they MAY opt to only expect IPv6 minimum MTU (1280) to avoid running into PMTUD implementation bugs or underlying link MTU mismatch problems.

### 6.13.4. Multiple links between nodes

If two nodes are connected via several links, the ACP SHOULD be established across every link, but it is possible to establish the ACP only on a sub-set of links. Having an ACP channel on every link has a number of advantages, for example it allows for a faster failover in case of link failure, and it reflects the physical topology more closely. Using a subset of links (for example, a single link), reduces resource consumption on the node, because state needs to be kept per ACP channel. The negotiation scheme explained in Section 6.6 allows the Decider (the node with the higher ACP address) to drop all but the desired ACP channels to the Follower - and the Follower will not re-try to build these secure channels from its side unless the Decider shows up with a previously unknown GRASP announcement (e.g., on a different link or with a different address announced in GRASP).

#### 6.13.5. ACP interfaces

The ACP VRF has conceptually two type of interfaces: The "ACP Loopback interface(s)" to which the ACP ULA address(es) are assigned and the "ACP virtual interfaces" that are mapped to the ACP secure channels.

##### 6.13.5.1. ACP loopback interfaces

For autonomous operations of the ACP, as described in Section 6 and Section 7, the ACP node uses the first address from the N bit ACP prefix (N = 128 - number of Vbits of the ACP address) assigned to the node. This address is assigned with an address prefix of N or larger to a loopback interface.

Other addresses from the prefix can be used by the ACP of the node as desired. The autonomous operations of the ACP does not require additional global scope IPv6 addresses, they are instead intended for ASA or non-autonomous functions. Non fully autonomic components of the ACP such as ACP connect interfaces (see Figure 16) may also introduce additional global scope IPv6 addresses on other types of interfaces into the ACP.

[RFC-Editor: please remove this paragraph: Note to reviewers: Please do not complain again about an obsolete RFC number in the following paragraph. The text should make it clear that the reference was chosen to indicate a particular point in time, but not to recommend/ use a particularly obsolete protocol spec.]

The use of loopback interfaces for global scope addresses is common operational configuration practice on routers, for example in IBGP connections since BGP4 (see [RFC1654]) or earlier. The ACP adopts and automates this operational practice.

A loopback interface for use with the ACP as described above is an interface behaving according to [RFC6724] Section 4., paragraph 2: Packets sent by the host of the node from the loopback interface behave as if they are looped back by the interface so that they look as if they originated from the loopback interface, are then received by the node and forwarded by it towards the destination.

The word loopback only indicates this behavior, but not the actual name of the interface type choosen in an actual implementation. A loopback interface for use with the ACP can be a virtual/software construct without any associated hardware, or it can be a hardware interface operating in loopback mode.

A loopback interface used for the ACP MUST NOT have connectivity to other nodes.

The following reviews the reasons for the choice of loopback addresses for ACP addresses is based on the IPv6 address architecture and common challenges:

1. IPv6 addresses are assigned to interfaces, not nodes. IPv6 continues the IPv4 model that a subnet prefix is associated with one link, see [RFC4291], Section 2.1.
2. IPv6 implementations commonly do not allow assignment of the same IPv6 global scope address in the same VRF to more than one interface.
3. Global scope addresses assigned to interfaces that are connecting to other nodes (external interfaces) may not be stable addresses for communications because any such interface could fail due to reasons external to the node. This could render the addresses assigned to that interface unusable.
4. If failure of the subnet does not result in bringing down the interface and making the addresses unusable, it could result in unreachability of the address because the shortest path to the node might go through one of the other nodes on the same subnet which could equally consider the subnet to be operational even though it is not.
5. Many OAM service implementations on routers cannot deal with more than one peer address, often because they do already expect that a single loopback address can be used, especially to provide a stable address under failure of external interfaces or links.
6. Even when an application supports multiple addresses to a peer, it can only use one address for a connection at a time with the most widely deployed transport protocols TCP and UDP. While [RFC6824] solves this problem, it is not widely adopted for router OAM services implementations.
7. To completely autonomously assign global scope addresses to subnets connecting to other nodes, it would be necessary for every node to have an amount of prefix address space in the order of the maximum number of subnets that the node could connect to and then the node would have to negotiate with adjacent nodes across those subnets whose address space to use for each subnet.
8. Using global scope addresses for subnets between nodes is unnecessary if those subnets only connect routers, such as ACP secure channels, because they can communicate to remote nodes via their global scope loopback addresses. Using global scope addresses for those extern subnets is therefore wasteful for the address space and also unnecessarily increasing the size of routing and forwarding tables, which especially for the ACP is highly undesirable because it should attempt to minimize the per-node overhead of the ACP VRF.

9. For all these reasons, the ACP addressing schemes do not consider ACP addresses for subnets connecting ACP nodes.

Note that [RFC8402] introduces the term Node-SID to refer to IGP prefix segments that identify a specific router, for example on a loopback interface. An ACP loopback address prefix may similarly be called an ACP Node Identifier.

#### 6.13.5.2. ACP virtual interfaces

Any ACP secure channel to another ACP node is mapped to ACP virtual interfaces in one of the following ways. This is independent of the chosen secure channel protocol (IPsec, DTLS or other future protocol - standards or non-standards).

Note that all the considerations described here are assuming point-to-point secure channel associations. Mapping multi-party secure channel associations such as [RFC6407] is out of scope.

##### 6.13.5.2.1. ACP point-to-point virtual interfaces

In this option, each ACP secure channel is mapped into a separate point-to-point ACP virtual interface. If a physical subnet has more than two ACP capable nodes (in the same domain), this implementation approach will lead to a full mesh of ACP virtual interfaces between them.

When the secure channel protocol determines a peer to be dead, this SHOULD result in indicating link breakage to trigger RPL DODAG repair, see Section 6.12.1.7.

##### 6.13.5.2.2. ACP multi-access virtual interfaces

In a more advanced implementation approach, the ACP will construct a single multi-access ACP virtual interface for all ACP secure channels to ACP capable nodes reachable across the same underlying (physical) subnet. IPv6 link-local multicast packets sent into an ACP multi-access virtual interface are replicated to every ACP secure channel mapped into the ACP multicast-access virtual interface. IPv6 unicast packets sent into an ACP multi-access virtual interface are sent to the ACP secure channel that belongs to the ACP neighbor that is the next-hop in the ACP forwarding table entry used to reach the packets destination address.

When the secure channel protocol determines a peer to be dead for a secure channel mapped into an ACP multi-access virtual interface, this SHOULD result in signaling breakage of that peer to RPL, so it can trigger RPL DODAG repair, see Section 6.12.1.7.

There is no requirement for all ACP nodes on the same multi-access subnet to use the same type of ACP virtual interface. This is purely a node local decision.

ACP nodes MUST perform standard IPv6 operations across ACP virtual interfaces including SLAAC (Stateless Address Auto-Configuration) - [RFC4862]) to assign their IPv6 link local address on the ACP virtual interface and ND (Neighbor Discovery - [RFC4861]) to discover which IPv6 link-local neighbor address belongs to which ACP secure channel mapped to the ACP virtual interface. This is independent of whether the ACP virtual interface is point-to-point or multi-access.

"Optimistic Duplicate Address Detection (DAD)" according to [RFC4429] is RECOMMENDED because the likelihood for duplicates between ACP nodes is highly improbable as long as the address can be formed from a globally unique local assigned identifier (e.g., EUI-48/EUI-64, see below).

ACP nodes MAY reduce the amount of link-local IPv6 multicast packets from ND by learning the IPv6 link-local neighbor address to ACP secure channel mapping from other messages such as the source address of IPv6 link-local multicast RPL messages - and therefore forego the need to send Neighbor Solicitation messages.

The ACP virtual interface IPv6 link local address can be derived from any appropriate local mechanism such as node local EUI-48 or EUI-64 ("EUI" stands for "Extended Unique Identifier"). It MUST NOT depend on something that is attackable from the Data-Plane such as the IPv6 link-local address of the underlying physical interface, which can be attacked by SLAAC, or parameters of the secure channel encapsulation header that may not be protected by the secure channel mechanism.

The link-layer address of an ACP virtual interface is the address used for the underlying interface across which the secure tunnels are built, typically Ethernet addresses. Because unicast IPv6 packets sent to an ACP virtual interface are not sent to a link-layer destination address but rather an ACP secure channel, the link-layer address fields SHOULD be ignored on reception and instead the ACP secure channel from which the message was received should be remembered.

Multi-access ACP virtual interfaces are preferable implementations when the underlying interface is a (broadcast) multi-access subnet because they do reflect the presence of the underlying multi-access subnet into the virtual interfaces of the ACP. This makes it for example simpler to build services with topology awareness inside the ACP VRF in the same way as they could have been built running natively on the multi-access interfaces.

Consider also the impact of point-to-point vs. multi-access virtual interface on the efficiency of flooding via link local multicasted messages:

Assume a LAN with three ACP neighbors, Alice, Bob and Carol. Alice's ACP GRASP wants to send a link-local GRASP multicast message to Bob and Carol. If Alice's ACP emulates the LAN as per-peer, point-to-point virtual interfaces, one to Bob and one to Carol, Alice's ACP GRASP will send two copies of multicast GRASP messages: One to Bob and one to Carol. If Alice's ACP emulates a LAN via a multipoint virtual interface, Alice's ACP GRASP will send one packet to that interface and the ACP multipoint virtual interface will replicate the packet to each secure channel, one to Bob, one to Carol. The result is the same. The difference happens when Bob and Carol receive their packet. If they use ACP point-to-point virtual interfaces, their GRASP instance would forward the packet from Alice to each other as part of the GRASP flooding procedure. These packets are unnecessary and would be discarded by GRASP on receipt as duplicates (by use of the GRASP Session ID). If Bob and Carol's ACP would emulate a multi-access virtual interface, then this would not happen, because GRASPs flooding procedure does not replicate back packets to the interface that they were received from.

Note that link-local GRASP multicast messages are not sent directly as IPv6 link-local multicast UDP messages into ACP virtual interfaces, but instead into ACP GRASP virtual interfaces, that are layered on top of ACP virtual interfaces to add TCP reliability to link-local multicast GRASP messages. Nevertheless, these ACP GRASP virtual interfaces perform the same replication of message and, therefore, result in the same impact on flooding. See Section 6.9.2 for more details.

RPL does support operations and correct routing table construction across non-broadcast multi-access (NBMA) subnets. This is common when using many radio technologies. When such NBMA subnets are used, they MUST NOT be represented as ACP multi-access virtual interfaces because the replication of IPv6 link-local multicast messages will not reach all NBMA subnet neighbors. In result, GRASP message flooding would fail. Instead, each ACP secure channel across such an interface MUST be represented as a ACP point-to-point virtual interface. See also Appendix A.9.4.

Care needs to be taken when creating multi-access ACP virtual interfaces across ACP secure channels between ACP nodes in different domains or routing subdomains. If for example future inter-domain ACP policies are defined as "peer-to-peer" policies, it is easier to create ACP point-to-point virtual interfaces for these inter-domain secure channels.

## 7. ACP support on L2 switches/ports (Normative)

### 7.1. Why (Benefits of ACP on L2 switches)

```
       ANrtr1 ------ ANswitch1 --- ANswitch2 ------- ANrtr2
                  .../   \                   \  ...
       ANrtrM ------      \                   ------- ANrtrN
                           ANswitchM ...
```

Figure 15: Topology with L2 ACP switches

Consider a large L2 LAN with ANrtr1...ANrtrN connected via some topology of L2 switches. Examples include large enterprise campus networks with an L2 core, IoT networks or broadband aggregation networks which often have even a multi-level L2 switched topology.

If the discovery protocol used for the ACP is operating at the subnet level, every ACP router will see all other ACP routers on the LAN as neighbors and a full mesh of ACP channels will be built. If some or all of the AN switches are autonomic with the same discovery protocol, then the full mesh would include those switches as well.

A full mesh of ACP connections can create fundamental scale challenges. The number of security associations of the secure channel protocols will likely not scale arbitrarily, especially when they leverage platform accelerated encryption/decryption. Likewise, any other ACP operations (such as routing) needs to scale to the number of direct ACP neighbors. An ACP router with just 4 physical interfaces might be deployed into a LAN with hundreds of neighbors connected via switches. Introducing such a new unpredictable scaling factor requirement makes it harder to support the ACP on arbitrary platforms and in arbitrary deployments.

Predictable scaling requirements for ACP neighbors can most easily be achieved if in topologies such as these, ACP capable L2 switches can ensure that discovery messages terminate on them so that neighboring ACP routers and switches will only find the physically connected ACP L2 switches as their candidate ACP neighbors. With such a discovery mechanism in place, the ACP and its security associations will only need to scale to the number of physical interfaces instead of a potentially much larger number of "LAN-connected" neighbors. And the ACP topology will follow directly the physical topology, something which can then also be leveraged in management operations or by ASAs.

In the example above, consider ANswitch1 and ANswitchM are ACP capable, and ANswitch2 is not ACP capable. The desired ACP topology is that ANrtr1 and ANrtrM only have an ACP connection to ANswitch1, and that ANswitch1, ANrtr2, ANrtrN have a full mesh of ACP connection amongst each other. ANswitch1 also has an ACP connection with ANswitchM and ANswitchM has ACP connections to anything else behind it.

## 7.2. How (per L2 port DULL GRASP)

To support ACP on L2 switches or L2 switched ports of an L3 device, it is necessary to make those L2 ports look like L3 interfaces for the ACP implementation. This primarily involves the creation of a separate DULL GRASP instance/domain on every such L2 port. Because GRASP has a dedicated link-local IPv6 multicast address (ALL_GRASP_NEIGHBORS), it is sufficient that all packets for this address are being extracted at the port level and passed to that DULL GRASP instance. Likewise the IPv6 link-local multicast packets sent by that DULL GRASP instance need to be sent only towards the L2 port for this DULL GRASP instance (instead of being flooded across all ports of the VLAN to which the port belongs).

When Ports/Interfaces across which the ACP is expected to operate in an ACP-aware L2-switch or L2/L3-switch/router are L2-bridged, packets for the ALL_GRASP_NEIGHBORS multicast address MUST never be forward between these ports. If MLD snooping is used, it MUST be prohibited from bridging packets for the ALL_GRASP_NEIGHBORS IPv6 multicast address.

On hybrid L2/L3 switches, multiple L2 ports are assigned to a single L3 VLAN interface. With the aforementioned changes for DULL GRASP, ACP can simply operate on the L3 VLAN interfaces, so no further (hardware) forwarding changes are required to make ACP operate on L2 ports. This is possible because the ACP secure channel protocols only use link-local IPv6 unicast packets, and these packets will be sent to the correct L2 port towards the peer by the VLAN logic of the device.

This is sufficient when p2p ACP virtual interfaces are established to every ACP peer. When it is desired to create multi-access ACP virtual interfaces (see Section 6.13.5.2.2), it is REQIURED not to coalesce all the ACP secure channels on the same L3 VLAN interface, but only all those on the same L2 port.

If VLAN tagging is used, then all the above described logic only applies to untagged GRASP packets. For the purpose of ACP neighbor discovery via GRASP, no VLAN tagged packets SHOULD be sent or received. In a hybrid L2/L3 switch, each VLAN would therefore only create ACP adjacencies across those ports where the VLAN is carried untagged.

In result, the simple logic is that ACP secure channels would operate over the same L3 interfaces that present a single flat bridged network across all routers, but because DULL GRASP is separated on a per-port basis, no full mesh of ACP secure channels is created, but only per-port ACP secure channels to per-port L2-adjacent ACP node neighbors.

For example, in the above picture, ANswitch1 would run separate DULL GRASP instances on its ports to ANrtr1, ANswitch2 and ANswitchI, even though all those three ports may be in the data plane in the same (V)LAN and perform L2 switching between these ports, ANswitch1 would perform ACP L3 routing between them.

The description in the previous paragraph was specifically meant to illustrate that on hybrid L3/L2 devices that are common in enterprise, IoT and broadband aggregation, there is only the GRASP packet extraction (by Ethernet address) and GRASP link-local multicast per L2-port packet injection that has to consider L2 ports at the hardware forwarding level. The remaining operations are purely ACP control plane and setup of secure channels across the L3 interface. This hopefully makes support for per-L2 port ACP on those hybrid devices easy.

In devices without such a mix of L2 port/interfaces and L3 interfaces (to terminate any transport layer connections), implementation details will differ. Logically most simply every L2 port is considered and used as a separate L3 subnet for all ACP operations. The fact that the ACP only requires IPv6 link-local unicast and multicast should make support for it on any type of L2 devices as simple as possible.

A generic issue with ACP in L2 switched networks is the interaction with the Spanning Tree Protocol. Without further L2 enhancements, the ACP would run only across the active STP topology and the ACP would be interrupted and re-converge with STP changes. Ideally, ACP peering SHOULD be built also across ports that are blocked in STP so that the ACP does not depend on STP and can continue to run unaffected across STP topology changes, where re-convergence can be quite slow. The above described simple implementation options are not sufficient to achieve this.

## 8. Support for Non-ACP Components (Normative)

### 8.1. ACP Connect

### 8.1.1. Non-ACP Controller / NMS system

The Autonomic Control Plane can be used by management systems, such as controllers or network management system (NMS) hosts (henceforth called simply "NMS hosts"), to connect to devices (or other type of nodes) through it. For this, an NMS host needs to have access to the ACP. The ACP is a self-protecting overlay network, which allows by default access only to trusted, autonomic systems. Therefore, a traditional, non-ACP NMS system does not have access to the ACP by default, such as any other external node.

If the NMS host is not autonomic, i.e., it does not support autonomic negotiation of the ACP, then it can be brought into the ACP by explicit configuration. To support connections to adjacent non-ACP nodes, an ACP node SHOULD support "ACP connect" (sometimes also called "autonomic connect"):

"ACP connect" is an interface level configured workaround for connection of trusted non-ACP nodes to the ACP. The ACP node on which ACP connect is configured is called an "ACP edge node". With ACP connect, the ACP is accessible from those non-ACP nodes (such as NOC systems) on such an interface without those non-ACP nodes having to support any ACP discovery or ACP channel setup. This is also called "native" access to the ACP because to those NOC systems the interface looks like a normal network interface without any ACP secure channel that is encapsulating the traffic.

```
                                 Data-Plane "native" (no ACP)
                                          .
+--------+       +----------------+       .        +-------------+
| ACP    |       |ACP Edge Node   |       .        |             |
| Node   |       |                |       v        |             |
|        |-------|...[ACP VRF]....+----------------|             |+
|        |   ^   |.               |                | NOC Device  ||
|        |   .   | .[Data-Plane]..+----------------| "NMS hosts" ||
|        |   .   |  [          ]  | .         ^    |             ||
+--------+   .   +----------------+  .        .    +-------------+|
             .                        .       .     +-------------+
             .                        .       .
          Data-Plane "native"         .   ACP "native" (unencrypted)
        + ACP auto-negotiated         .   "ACP connect subnet"
          and encrypted               .
                                    ACP connect interface
                                    e.g., "VRF ACP native" (config)
```

Figure 16: ACP connect

ACP connect has security consequences: All systems and processes connected via ACP connect have access to all ACP nodes on the entire ACP, without further authentication. Thus, the ACP connect interface and NOC systems connected to it needs to be physically controlled/ secured. For this reason the mechanisms described here do explicitly not include options to allow for a non-ACP router to be connected across an ACP connect interface and addresses behind such a router routed inside the ACP.

Physical controlled/secured means that attackers can gain no access to the physical device hosting the ACP Edge Node, the physical interfaces and links providing the ACP connect link nor the physical devices hosting the NOC Device. In a simple case, ACP Edge node and NOC Device are co-located in an access controlled room, such as a NOC, to which attackers cannot gain physical access.

An ACP connect interface provides exclusively access to only the ACP. This is likely insufficient for many NMS hosts. Instead, they would require a second "Data-Plane" interface outside the ACP for connections between the NMS host and administrators, or Internet based services, or for direct access to the Data-Plane. The document "Using Autonomic Control Plane for Stable Connectivity of Network OAM" [RFC8368] explains in more detail how the ACP can be integrated in a mixed NOC environment.

An ACP connect interface SHOULD use an IPv6 address/prefix from the ACP Manual Addressing Sub-Scheme (Section 6.11.4), letting the operator configure for example only the Subnet-ID and having the node automatically assign the remaining part of the prefix/address. It SHOULD NOT use a prefix that is also routed outside the ACP so that the addresses clearly indicate whether it is used inside the ACP or not.

The prefix of ACP connect subnets MUST be distributed by the ACP edge node into the ACP routing protocol RPL. The NMS hosts MUST connect to prefixes in the ACP routing table via its ACP connect interface. In the simple case where the ACP uses only one ULA prefix and all ACP connect subnets have prefixes covered by that ULA prefix, NMS hosts can rely on [RFC6724] to determine longest match prefix routes towards its different interfaces, ACP and Data-Plane. With RFC6724, The NMS host will select the ACP connect interface for all addresses in the ACP because any ACP destination address is longest matched by the address on the ACP connect interface. If the NMS hosts ACP connect interface uses another prefix or if the ACP uses multiple ULA prefixes, then the NMS hosts require (static) routes towards the ACP interface for these prefixes.

When an ACP Edge node receives a packet from an ACP connect interface, the ACP Edge node MUST only forward the packet into the ACP if the packet has an IPv6 source address from that interface (this is sometimes called "RPF filtering"). This filtering rule MAY be changed through administrative measures. The more any such administrative action enable reachability of non ACP nodes to the ACP, the more this may cause security issues.

To limit the security impact of ACP connect, nodes supporting it SHOULD implement a security mechanism to allow configuration/use of ACP connect interfaces only on nodes explicitly targeted to be deployed with it (those in physically secure locations such as a NOC). For example, the registrar could disable the ability to enable ACP connect on devices during enrollment and that property could only be changed through re-enrollment. See also Appendix A.9.5.

ACP Edge nodes SHOULD have a configurable option to prohibit packets with RPI headers (see Section 6.12.1.13 across an ACP connect interface. These headers are outside the scope of the RPL profile in this specification but may be used in future extensions of this specification.

### 8.1.2. Software Components

The previous section assumed that ACP Edge node and NOC devices are separate physical devices and the ACP connect interface is a physical network connection. This section discusses the implication when these components are instead software components running on a single physical device.

The ACP connect mechanism cannot only be used to connect physically external systems (NMS hosts) to the ACP but also other applications, containers or virtual machines. In fact, one possible way to eliminate the security issue of the external ACP connect interface is to collocate an ACP edge node and an NMS host by making one a virtual machine or container inside the other; and therefore converting the unprotected external ACP subnet into an internal virtual subnet in a single device. This would ultimately result in a fully ACP enabled NMS host with minimum impact to the NMS hosts software architecture. This approach is not limited to NMS hosts but could equally be applied to devices consisting of one or more VNF (virtual network functions): An internal virtual subnet connecting out-of-band management interfaces of the VNFs to an ACP edge router VNF.

The core requirement is that the software components need to have a network stack that permits access to the ACP and optionally also the Data-Plane. Like in the physical setup for NMS hosts this can be realized via two internal virtual subnets. One that is connecting to the ACP (which could be a container or virtual machine by itself), and one (or more) connecting into the Data-Plane.

This "internal" use of ACP connect approach should not be considered to be a "workaround" because in this case it is possible to build a correct security model: It is not necessary to rely on unprovable external physical security mechanisms as in the case of external NMS hosts. Instead, the orchestration of the ACP, the virtual subnets and the software components can be done by trusted software that could be considered to be part of the ANI (or even an extended ACP). This software component is responsible for ensuring that only trusted software components will get access to that virtual subnet and that only even more trusted software components will get access to both the ACP virtual subnet and the Data-Plane (because those ACP users could leak traffic between ACP and Data-Plane). This trust could be established for example through cryptographic means such as signed software packages.

### 8.1.3. Auto Configuration

ACP edge nodes, NMS hosts and software components that as described in the previous section are meant to be composed via virtual interfaces SHOULD support on the ACP connect subnet StateLess Address Autoconfiguration (SLAAC - [RFC4862]) and route auto configuration according to [RFC4191].

The ACP edge node acts as the router towards the ACP on the ACP connect subnet, providing the (auto-)configured prefix for the ACP connect subnet and (auto-)configured routes into the ACP to NMS hosts and/or software components.

The ACP edge node uses the Route Information Option (RIO) of RFC4191 to announce aggregated prefixes for address prefixes used in the ACP (with normal RIO lifetimes. In addition, the ACP edge node also uses a RIO to announce the default route (::/0) with a lifetime of 0.

These RIOs allow to connect Type C hosts to the ACP via an ACP connect subnet on one interface and another network (Data Plane / NMS network) on the same or another interface of the Type C host, relying on other routers than the ACP edge node. The RIOs ensure that these hosts will only route the prefixes used in the ACP to the ACP edge node.

Type A/B host ignore the RIOs and will consider the ACP node to be their default router for all destination. This is sufficient when type A/B hosts only need to connect to the ACP but not to other networks. Attaching Type A/B hosts to both the ACP and other networks, requires either explicit ACP prefix route configuration on the Type A/B hosts or the combined ACP/Data-Plane interface on the ACP edge node, see Section 8.1.4.

Aggregated prefix means that the ACP edge node needs to only announce the /48 ULA prefixes used in the ACP but none of the actual /64 (Manual Addressing Sub-Scheme), /127 (ACP Zone Addressing Sub-Scheme), /112 or /120 (Vlong Addressing Sub-Scheme) routes of actual ACP nodes. If ACP interfaces are configured with non ULA prefixes, then those prefixes cannot be aggregated without further configured policy on the ACP edge node. This explains the above recommendation to use ACP ULA prefix covered prefixes for ACP connect interfaces: They allow for a shorter list of prefixes to be signaled via RFC4191 to NMS hosts and software components.

The ACP edge nodes that have a Vlong ACP address MAY allocate a subset of their /112 or /120 address prefix to ACP connect interface(s) to eliminate the need to non-autonomically configure/provision the address prefixes for such ACP connect interfaces.

### 8.1.4. Combined ACP/Data-Plane Interface (VRF Select)

```
                   Combined ACP and Data-Plane interface
                                          .
  +--------+       +--------------------+    .   +--------------+
  | ACP    |       |ACP Edge No         |    .   | NMS Host(s)  |
  | Node   |       |                    |    .   | / Software   |
  |        |       |  [ACP  ].          |    .   |              |+
  |        |       | .[VRF  ] .[VRF   ] |    v   | "ACP address"||
  |        +-------+.         .[Select].+--------+ "Date Plane   ||
  |        |   ^   | .[Data ].          |        |  Address(es)"||
  |        |   .   |  [Plane]           |        |              ||
  |        |   .   |  [     ]           |        +--------------+|
  +--------+   .   +--------------------+         +--------------+
               .
     Data-Plane "native" and + ACP auto-negotiated/encrypted
```

Figure 17: VRF select

Using two physical and/or virtual subnets (and therefore interfaces) into NMS Hosts (as per Section 8.1.1) or Software (as per Section 8.1.2) may be seen as additional complexity, for example with legacy NMS Hosts that support only one IP interface, or it may be insufficient to support [RFC4191] Type A or B host (see Section 8.1.3).

To provide a single subnet into both ACP and Data-Plane, the ACP Edge node needs to de-multiplex packets from NMS hosts into ACP VRF and Data-Plane.  This is sometimes called "VRF select".  If the ACP VRF has no overlapping IPv6 addresses with the Data-Plane (it should have no overlapping addresses), then this function can use the IPv6 Destination address.  The problem is Source Address Selection on the NMS Host(s) according to RFC6724.

Consider the simple case: The ACP uses only one ULA prefix, the ACP IPv6 prefix for the Combined ACP and Data-Plane interface is covered by that ULA prefix.  The ACP edge node announces both the ACP IPv6 prefix and one (or more) prefixes for the Data-Plane.  Without further policy configurations on the NMS Host(s), it may select its ACP address as a source address for Data-Plane ULA destinations because of Rule 8 of RFC6724.  The ACP edge node can pass on the packet to the Data-Plane, but the ACP source address should not be used for Data-Plane traffic, and return traffic may fail.

If the ACP carries multiple ULA prefixes or non-ULA ACP connect prefixes, then the correct source address selection becomes even more problematic.

With separate ACP connect and Data-Plane subnets and RFC4191 prefix announcements that are to be routed across the ACP connect interface, RFC6724 source address selection Rule 5 (use address of outgoing interface) will be used, so that above problems do not occur, even in more complex cases of multiple ULA and non-ULA prefixes in the ACP routing table.

To achieve the same behavior with a Combined ACP and Data-Plane interface, the ACP Edge Node needs to behave as two separate routers on the interface: One link-local IPv6 address/router for its ACP reachability, and one link-local IPv6 address/router for its Data-Plane reachability.  The Router Advertisements for both are as described above (Section 8.1.3): For the ACP, the ACP prefix is announced together with RFC4191 option for the prefixes routed across the ACP and lifetime=0 to disqualify this next-hop as a default router.  For the Data-Plane, the Data-Plane prefix(es) are announced together with whatever dafault router parameters are used for the Data-Plane.

In result, RFC6724 source address selection Rule 5.5 may result in the same correct source address selection behavior of NMS hosts without further configuration on it as the separate ACP connect and Data-Plane interfaces. As described in the text for Rule 5.5, this is only a MAY, because IPv6 hosts are not required to track next-hop information. If an NMS Host does not do this, then separate ACP connect and Data-Plane interfaces are the preferable method of attachment. Hosts implementing [RFC8028] should (instead of may) implement [RFC6724] Rule 5.5, so it is preferred for hosts to support [RFC8028].

ACP edge nodes MAY support the Combined ACP and Data-Plane interface.

#### 8.1.5. Use of GRASP

GRASP can and should be possible to use across ACP connect interfaces, especially in the architectural correct solution when it is used as a mechanism to connect Software (e.g., ASA or legacy NMS applications) to the ACP.

Given how the ACP is the security and transport substrate for GRASP, the requirements for devices connected via ACP connect is that those are equivalently (if not better) secured against attacks than ACP nodes that do not use ACP connect and run only software that is equally (if not better) protected, known (or trusted) not to be malicious and accordingly designed to isolate access to the ACP against external equipment.

The difference in security is that cryptographic security of the ACP secure channel is replaced by required physical security/control of the network connection between an ACP edge node and the NMS or other host reachable via the ACP connect interface. See Section 8.1.1.

When using "Combined ACP and Data-Plane Interfaces", care has to be taken that only GRASP messages intended for the ACP GRASP domain received from Software or NMS Hosts are forwarded by ACP edge nodes. Currently there is no definition for a GRASP security and transport substrate beside the ACP, so there is no definition how such Software/NMS Host could participate in two separate GRASP Domains across the same subnet (ACP and Data-Plane domains). At current it is assumed that all GRASP packets on a Combined ACP and Data-Plane interface belong to the GRASP ACP Domain. They SHOULD all use the ACP IPv6 addresses of the Software/NMS Hosts. The link-local IPv6 addresses of Software/NMS Hosts (used for GRASP M_DISCOVERY and M_FLOOD messages) are also assumed to belong to the ACP address space.

### 8.2. Connecting ACP islands over Non-ACP L3 networks (Remote ACP neighbors)

Not all nodes in a network may support the ACP. If non-ACP Layer-2 devices are between ACP nodes, the ACP will work across it since it is IP based. However, the autonomic discovery of ACP neighbors via DULL GRASP is only intended to work across L2 connections, so it is not sufficient to autonomically create ACP connections across non-ACP Layer-3 devices.

#### 8.2.1. Configured Remote ACP neighbor

On the ACP node, remote ACP neighbors are configured explicitly. The parameters of such a "connection" are described in the following ABNF.

```
connection = [ method , local-addr, remote-addr, ?pmtu ]
method =  [ "IKEv2", ?port ]
method =/ [ "DTLS",    port ]
local-addr  = [ address , ?vrf  ]
remote-addr = [ address ]
address = ("any" | ipv4-address | ipv6-address )
vrf = tstr ; Name of a VRF on this node with local-address
```

Figure 18: Parameters for remote ACP neighbors

Explicit configuration of a remote-peer according to this ABNF provides all the information to build a secure channel without requiring a tunnel to that peer and running DULL GRASP inside of it.

The configuration includes the parameters otherwise signaled via DULL GRASP: local address, remote (peer) locator and method. The differences over DULL GRASP local neighbor discovery and secure channel creation are as follows:

* The local and remote address can be IPv4 or IPv6 and are typically global scope addresses.
* The VRF across which the connection is built (and in which local-addr exists) can to be specified. If vrf is not specified, it is the default VRF on the node. In DULL GRASP the VRF is implied by the interface across which DULL GRASP operates.
* If local address is "any", the local address used when initiating a secure channel connection is decided by source address selection ([RFC6724] for IPv6). As a responder, the connection listens on all addresses of the node in the selected VRF.
* Configuration of port is only required for methods where no defaults exist (e.g., "DTLS").

* If remote address is "any", the connection is only a responder. It is a "hub" that can be used by multiple remote peers to connect simultaneously - without having to know or configure their addresses. Example: Hub site for remote "spoke" sites reachable over the Internet.
* Pmtu should be configurable to overcome issues/limitations of Path MTU Discovery (PMTUD).
* IKEv2/IPsec to remote peers should support the optional NAT Traversal (NAT-T) procedures.

#### 8.2.2. Tunneled Remote ACP Neighbor

An IPinIP, GRE or other form of pre-existing tunnel is configured between two remote ACP peers and the virtual interfaces representing the tunnel are configured for "ACP enable". This will enable IPv6 link local addresses and DULL on this tunnel. In result, the tunnel is used for normal "L2 adjacent" candidate ACP neighbor discovery with DULL and secure channel setup procedures described in this document.

Tunneled Remote ACP Neighbor requires two encapsulations: the configured tunnel and the secure channel inside of that tunnel. This makes it in general less desirable than Configured Remote ACP Neighbor. Benefits of tunnels are that it may be easier to implement because there is no change to the ACP functionality - just running it over a virtual (tunnel) interface instead of only native interfaces. The tunnel itself may also provide PMTUD while the secure channel method may not. Or the tunnel mechanism is permitted/possible through some firewall while the secure channel method may not.

Tunneling using an insecure tunnel encapsulation increases on average the risk of a MITM downgrade attack somewhere along the underlay path that blocks all but the most easily attacked ACP secure channel option. ACP nodes supporting tunneled remote ACP Neighbors SHOULD support configuration on such tunnel interfaces to restrict or explicitly select the available ACP secure channel protocols (if the ACP node supports more than one ACP secure channel protocol in the first place).

#### 8.2.3. Summary

Configured/Tunneled Remote ACP neighbors are less "indestructible" than L2 adjacent ACP neighbors based on link local addressing, since they depend on more correct Data-Plane operations, such as routing and global addressing.

Nevertheless, these options may be crucial to incrementally deploy the ACP, especially if it is meant to connect islands across the Internet. Implementations SHOULD support at least Tunneled Remote ACP Neighbors via GRE tunnels - which is likely the most common router-to-router tunneling protocol in use today.

# 9. ACP Operations (Informative)

The following sections document important operational aspects of the ACP. They are not normative because they do not impact the interoperability between components of the ACP, but they include recommendations/requirements for the internal operational model beneficial or necessary to achieve the desired use-case benefits of the ACP (see Section 3).

* Section 9.1 describes recommended operator diagnostics capabilities of ACP nodes.
* Section 9.2 describes high level how an ACP registrar needs to work, what its configuration parameters are and specific issues impacting the choices of deployment design due to renewal and revocation issues. It describes a model where ACP Registrars have their own sub-CA to provide the most distributed deployment option for ACP Registrars, and it describes considerations for centralized policy control of ACP Registrar operations.
* Section 9.3 describes suggested ACP node behavior and operational interfaces (configuration options) to manage the ACP in so-called greenfield devices (previously unconfigured) and brownfield devices (preconfigured).

The recommendations and suggestions of this chapter were derived from operational experience gained with a commercially available pre-standard ACP implementation.

## 9.1. ACP (and BRSKI) Diagnostics

Even though ACP and ANI in general are taking out many manual configuration mistakes through their automation, it is important to provide good diagnostics for them.

Basic standardized diagnostics would require support for (yang) models representing the complete (auto-)configuration and operational state of all components: GRASP, ACP and the infrastructure used by them: TLS/DTLS, IPsec, certificates, TA, time, VRF and so on. While necessary, this is not sufficient:

Simply representing the state of components does not allow operators to quickly take action - unless they do understand how to interpret the data, and that can mean a requirement for deep understanding of all components and how they interact in the ACP/ANI.

Diagnostic supports should help to quickly answer the questions operators are expected to ask, such as "is the ACP working correctly?", or "why is there no ACP connection to a known neighboring node?"

In current network management approaches, the logic to answer these questions is most often built as centralized diagnostics software that leverages the above mentioned data models. While this approach is feasible for components utilizing the ANI, it is not sufficient to diagnose the ANI itself:

* Developing the logic to identify common issues requires operational experience with the components of the ANI. Letting each management system define its own analysis is inefficient.
* When the ANI is not operating correctly, it may not be possible to run diagnostics from remote because of missing connectivity. The ANI should therefore have diagnostic capabilities available locally on the nodes themselves.
* Certain operations are difficult or impossible to monitor in real-time, such as initial bootstrap issues in a network location where no capabilities exist to attach local diagnostics. Therefore, it is important to also define means of capturing (logging) diagnostics locally for later retrieval. Ideally, these captures are also non-volatile so that they can survive extended power-off conditions - for example when a device that fails to be brought up zero-touch is being sent back for diagnostics at a more appropriate location.

The simplest form of diagnostics answering questions such as the above is to represent the relevant information sequentially in dependency order, so that the first non-expected/non-operational item is the most likely root cause. Or just log/highlight that item. For example:

Q: Is ACP operational to accept neighbor connections:

* Check if any potentially necessary configuration to make ACP/ANI operational are correct (see Section 9.3 for a discussion of such commands).
* Does the system time look reasonable, or could it be the default system time after clock chip battery failure (certificate checks depend on reasonable notion of time)?.

* Does the node have keying material - domain certificate, TA certificates, ...>
* If no keying material and ANI is supported/enabled, check the state of BRSKI (not detailed in this example).
* Check the validity of the domain certificate:
  - Does the certificate validate against the TA?
  - Has it been revoked?
  - Was the last scheduled attempt to retrieve a CRL successful (e.g., do we know that our CRL information is up to date).
  - Is the certificate valid: validity start time in the past, expiration time in the future?
  - Does the certificate have a correctly formatted acp-node-name field?
* Was the ACP VRF successfully created?
* Is ACP enabled on one or more interfaces that are up and running?

If all this looks good, the ACP should be running locally "fine" - but we did not check any ACP neighbor relationships.

Question: why does the node not create a working ACP connection to a neighbor on an interface?

* Is the interface physically up? Does it have an IPv6 link-local address?
* Is it enabled for ACP?
* Do we successfully send DULL GRASP messages to the interface (link layer errors)?
* Do we receive DULL GRASP messages on the interface? If not, some intervening L2 equipment performing bad MLD snooping could have caused problems. Provide e.g., diagnostics of the MLD querier IPv6 and MAC address.
* Do we see the ACP objective in any DULL GRASP message from that interface? Diagnose the supported secure channel methods.
* Do we know the MAC address of the neighbor with the ACP objective? If not, diagnose SLAAC/ND state.
* When did we last attempt to build an ACP secure channel to the neighbor?
* If it failed, why:
  - Did the neighbor close the connection on us or did we close the connection on it because the domain certificate membership failed?
  - If the neighbor closed the connection on us, provide any error diagnostics from the secure channel protocol.
  - If we failed the attempt, display our local reason:
    o There was no common secure channel protocol supported by the two neighbors (this could not happen on nodes supporting this specification because it mandates common support for IPsec).

- o The ACP certificate membership check (Section 6.2.3) fails:
  - + The neighbor's certificate is not signed directly or indirectly by one of the nodes TA. Provide diagnostics which TA it has (can identify whom the device belongs to).
  - + The neighbor's certificate does not have the same domain (or no domain at all). Diagnose domain-name and potentially other cert info.
  - + The neighbor's certificate has been revoked or could not be authenticated by OCSP.
  - + The neighbor's certificate has expired - or is not yet valid.
- \- Any other connection issues in e.g., IKEv2 / IPsec, DTLS?.

Question: Is the ACP operating correctly across its secure channels?

* Are there one or more active ACP neighbors with secure channels?
* Is the RPL routing protocol for the ACP running?
* Is there a default route to the root in the ACP routing table?
* Is there for each direct ACP neighbor not reachable over the ACP virtual interface to the root a route in the ACP routing table?
* Is ACP GRASP running?
* Is at least one SRV.est objective cached (to support certificate renewal)?
* Is there at least one BRSKI registrar objective cached (in case BRSKI is supported)
* Is BRSKI proxy operating normally on all interfaces where ACP is operating?
* ...

These lists are not necessarily complete, but illustrate the principle and show that there are variety of issues ranging from normal operational causes (a neighbor in another ACP domain) over problems in the credentials management (certificate lifetimes), explicit security actions (revocation) or unexpected connectivity issues (intervening L2 equipment).

The items so far are illustrating how the ANI operations can be diagnosed with passive observation of the operational state of its components including historic/cached/counted events. This is not necessary sufficient to provide good enough diagnostics overall:

The components of ACP and BRSKI are designed with security in mind but they do not attempt to provide diagnostics for building the network itself. Consider two examples:

1. BRSKI does not allow for a neighboring device to identify the pledges IDevID certificate. Only the selected BRSKI registrar can do this, but it may be difficult to disseminate information about undesired pledges from those BRSKI registrars to locations/ nodes where information about those pledges is desired.
2. LLDP disseminates information about nodes to their immediate neighbors, such as node model/type/software and interface name/ number of the connection. This information is often helpful or even necessary in network diagnostics. It can equally be considered to be too insecure to make this information available unprotected to all possible neighbors.

An "interested adjacent party" can always determine the IDevID certificate of a BRSKI pledge by behaving like a BRSKI proxy/ registrar. Therefore, the IDevID certificate of a BRSKI pledge is not meant to be protected - it just has to be queried and is not signaled unsolicited (as it would be in LLDP) so that other observers on the same subnet can determine who is an "interested adjacent party".

#### 9.1.1. Secure Channel Peer diagnostics

When using mutual certificate authentication, the TA certificate is not required to be signaled explicitly because its hash is sufficient for certificate chain validation. In the case of ACP secure channel setup this leads to limited diagnostics when authentication fails because of TA mismatch. For this reason, Section 6.8.2 recommends to also include the TA certificate in the secure channel signaling. This should be possible to do without protocol modifications in the security association protocols used by the ACP. For example, while [RFC7296] does not mention this, it also does not prohibit it.

One common deployment use case where the diagnostic through the signaled TA of a candidate peer is very helpful are multi-tenant environments such as office buildings, where different tenants run their own networks and ACPs. Each tenant is given supposedly disjoint L2 connectivity through the building infrastructure. In these environments there are various common errors through which a device may receive L2 connectivity into the wrong tenants network.

While the ACP itself is not impact by this, the Data-Plane to be built later may be impacted. Therefore, it is important to be able to diagnose such undesirable connectivity from the ACP so that any autonomic or non-autonomic mechanisms to configure the Data-Plane can accordingly treat such interfaces. The information in the TA of the peer can then ease troubleshooting of such issues.

Another example case is the intended or accidental re-activation of equipment whose TA certificate has long expired, such as redundant gear taken from storage after years.

A third example case is when in a mergers & acquisition case ACP nodes have not been correctly provisioned with the mutual TA of previously disjoint ACP. This is assuming that the ACP domain names where already aligned so that the ACP domain membership check is only failing on the TA.

A fourth example case is when multiple registrars where set up for the same ACP but without correctly setting up the same TA. For example, when registrars support to also be CA themselves but are misconfigured to become TA instead of intermediate CA.

## 9.2. ACP Registrars

As described in Section 6.11.7, the ACP addressing mechanism is designed to enable lightweight, distributed and uncoordinated ACP registrars that are providing ACP address prefixes to candidate ACP nodes by enrolling them with an ACP certificate into an ACP domain via any appropriate mechanism/protocol, automated or not.

This section discusses informatively more details and options for ACP registrars.

### 9.2.1. Registrar interactions

This section summarizes and discusses the interactions with other entities required by an ACP registrar.

In a simple instance of an ACP network, no central NOC component beside a TA is required. Typically, this is a root CA. One or more uncoordinated acting ACP registrar can be set up, performing the following interactions:

To orchestrate enrolling a candidate ACP node autonomically, the ACP registrar can rely on the ACP and use Proxies to reach the candidate ACP node, therefore allowing minimum pre-existing (auto-)configured network services on the candidate ACP node. BRSKI defines the BRSKI proxy, a design that can be adopted for various protocols that Pledges/candidate ACP nodes could want to use, for example BRSKI over CoAP (Constrained Application Protocol), or proxying of NETCONF.

To reach a TA that has no ACP connectivity, the ACP registrar would use the Data-Plane. ACP and Data-Plane in an ACP registrar could (and by default should be) completely isolated from each other at the network level. Only applications such as the ACP registrar would need the ability for their transport stacks to access both.

In non-autonomic enrollment options, the Data-Plane between a ACP registrar and the candidate ACP node needs to be configured first. This includes the ACP registrar and the candidate ACP node. Then any appropriate set of protocols can be used between ACP registrar and candidate ACP node to discover the other side, and then connect and enroll (configure) the candidate ACP node with an ACP certificate. NETCONF ZeroTouch ([RFC8572]) is an example protocol that could be used for this. BRSKI using optional discovery mechanisms is equally a possibility for candidate ACP nodes attempting to be enrolled across non-ACP networks, such as the Internet.

When candidate ACP nodes have secure bootstrap, such as BRSKI Pledges, they will not trust to be configured/enrolled across the network, unless being presented with a voucher (see [RFC8366]) authorizing the network to take possession of the node. An ACP registrar will then need a method to retrieve such a voucher, either offline, or online from a MASA (Manufacturer Authorized Signing Authority). BRSKI and NETCONF ZeroTouch are two protocols that include capabilities to present the voucher to the candidate ACP node.

An ACP registrar could operate EST for ACP certificate renewal and/or act as a CRL Distribution point. A node performing these services does not need to support performing (initial) enrollment, but it does require the same above described connectivity as an ACP registrar: via the ACP to ACP nodes and via the Data-Plane to the TA and other sources of CRL information.

### 9.2.2. Registrar Parameter

The interactions of an ACP registrar outlined Section 6.11.7 and Section 9.2.1 above depend on the following parameters:

* A URL to the TA and credentials so that the ACP registrar can let the TA sign candidate ACP node certificates.
* The ACP domain-name.
* The Registrar-ID to use. This could default to a MAC address of the ACP registrar.

* For recovery, the next-useable Node-IDs for zone (Zone-ID=0) sub-addressing scheme, for Vlong /112 and for Vlong /120 sub-addressing scheme. These IDs would only need to be provisioned after recovering from a crash. Some other mechanism would be required to remember these IDs in a backup location or to recover them from the set of currently known ACP nodes.
* Policies if candidate ACP nodes should receive a domain certificate or not, for example based on the devices IDevID certificate as in BRSKI. The ACP registrar may have a whitelist or blacklist of devices [X.520] "serialNumbers" attribute in the subject field distinguished name encoding from their IDevID certificate.
* Policies what type of address prefix to assign to a candidate ACP devices, based on likely the same information.
* For BRSKI or other mechanisms using vouchers: Parameters to determine how to retrieve vouchers for specific type of secure bootstrap candidate ACP nodes (such as MASA URLs), unless this information is automatically learned such as from the IDevID certificate of candidate ACP nodes (as defined in BRSKI).

#### 9.2.3. Certificate renewal and limitations

When an ACP node renews/rekeys its certificate, it may end up doing so via a different registrar (e.g., EST server) than the one it originally received its ACP certificate from, for example because that original ACP registrar is gone. The ACP registrar through which the renewal/rekeying is performed would by default trust the acp-node-name from the ACP nodes current ACP certificate and maintain this information so that the ACP node maintains its ACP address prefix. In EST renewal/rekeying, the ACP nodes current ACP certificate is signaled during the TLS handshake.

This simple scenario has two limitations:

1. The ACP registrars cannot directly assign certificates to nodes and therefore needs an "online" connection to the TA.
2. Recovery from a compromised ACP registrar is difficult. When an ACP registrar is compromised, it can insert for example a conflicting acp-node-name and create thereby an attack against other ACP nodes through the ACP routing protocol.

Even when such a malicious ACP registrar is detected, resolving the problem may be difficult because it would require identifying all the wrong ACP certificates assigned via the ACP registrar after it was compromised. And without additional centralized tracking of assigned certificates there is no way to do this.

#### 9.2.4. ACP Registrars with sub-CA

In situations, where either of the above two limitations are an issue, ACP registrars could also be sub-CAs. This removes the need for connectivity to a TA whenever an ACP node is enrolled, and reduces the need for connectivity of such an ACP registrar to a TA to only those times when it needs to renew its own certificate. The ACP registrar would also now use its own (sub-CA) certificate to enroll and sign the ACP nodes certificates, and therefore it is only necessary to revoke a compromised ACP registrars sub-CA certificate. Alternatively one can let it expire and not renew it, when the certificate of the sub-CA is appropriately short-lived.

As the ACP domain membership check verifies a peer ACP node's ACP certificate trust chain, it will also verify the signing certificate which is the compromised/revoked sub-CA certificate. Therefore, ACP domain membership for an ACP node enrolled from a compromised and discovered ACP registrar will fail.

ACP nodes enrolled by a compromised ACP registrar would automatically fail to establish ACP channels and ACP domain certificate renewal via EST and therefore revert to their role as a candidate ACP members and attempt to get a new ACP certificate from an ACP registrar - for example, via BRSKI. In result, ACP registrars that have an associated sub-CA makes isolating and resolving issues with compromised registrars easier.

Note that ACP registrars with sub-CA functionality also can control the lifetime of ACP certificates easier and therefore also be used as a tool to introduce short lived certificates and not rely on CRL, whereas the certificates for the sub-CAs themselves could be longer lived and subject to CRL.

#### 9.2.5. Centralized Policy Control

When using multiple, uncoordinated ACP registrars, several advanced operations are potentially more complex than with a single, resilient policy control backend, for example including but not limited to:

* Which candidate ACP node is permitted or not permitted into an ACP domain. This may not be a decision to be taken upfront, so that a policy per "serialNumber" attribute in the subject field distinguished name encoding can be loaded into every ACP registrar. Instead, it may better be decided in real-time including potentially a human decision in a NOC.
* Tracking of all enrolled ACP nodes and their certificate information. For example, in support of revoking individual ACP nodes certificates.

* More flexible policies what type of address prefix or even what specific address prefix to assign to a candidate ACP node.

These and other operations could be introduced more easily by introducing a centralized Policy Management System (PMS) and modifying ACP registrar behavior so that it queries the PMS for any policy decision occurring during the candidate ACP node enrollment process and/or the ACP node certificate renewal process. For example, which ACP address prefix to assign. Likewise the ACP registrar would report any relevant state change information to the PMS as well, for example when a certificate was successfully enrolled onto a candidate ACP node.

### 9.3. Enabling and disabling ACP/ANI

Both ACP and BRSKI require interfaces to be operational enough to support sending/receiving their packets. In node types where interfaces are by default (e.g., without operator configuration) enabled, such as most L2 switches, this would be less of a change in behavior than in most L3 devices (e.g. routers), where interfaces are by default disabled. In almost all network devices it is common though for configuration to change interfaces to a physically disabled state and that would break the ACP.

In this section, we discuss a suggested operational model to enable/ disable interfaces and nodes for ACP/ANI in a way that minimizes the risk of operator action to break the ACP in this way, and that also minimizes operator surprise when ACP/ANI becomes supported in node software.

#### 9.3.1. Filtering for non-ACP/ANI packets

Whenever this document refers to enabling an interface for ACP (or BRSKI), it only requires to permit the interface to send/receive packets necessary to operate ACP (or BRSKI) - but not any other Data-Plane packets. Unless the Data-Plane is explicitly configured/ enabled, all packets not required for ACP/BRSKI should be filtered on input and output:

Both BRSKI and ACP require link-local only IPv6 operations on interfaces and DULL GRASP. IPv6 link-local operations means the minimum signaling to auto-assign an IPv6 link-local address and talk to neighbors via their link-local address: SLAAC (Stateless Address Auto-Configuration - [RFC4862]) and ND (Neighbor Discovery - [RFC4861]). When the device is a BRSKI pledge, it may also require TCP/TLS connections to BRSKI proxies on the interface. When the device has keying material, and the ACP is running, it requires DULL GRASP packets and packets necessary for the secure-channel mechanism it supports, e.g., IKEv2 and IPsec ESP packets or DTLS packets to the IPv6 link-local address of an ACP neighbor on the interface. It also requires TCP/TLS packets for its BRSKI proxy functionality, if it does support BRSKI.

#### 9.3.2. Admin Down State

Interfaces on most network equipment have at least two states: "up" and "down". These may have product specific names. "down" for example could be called "shutdown" and "up" could be called "no shutdown". The "down" state disables all interface operations down to the physical level. The "up" state enables the interface enough for all possible L2/L3 services to operate on top of it and it may also auto-enable some subset of them. More commonly, the operations of various L2/L3 services is controlled via additional node-wide or interface level options, but they all become only active when the interface is not "down". Therefore, an easy way to ensure that all L2/L3 operations on an interface are inactive is to put the interface into "down" state. The fact that this also physically shuts down the interface is in many cases just a side effect, but it may be important in other cases (see below, Section 9.3.2.2).

One of the common problems of remote management is for the operator or SDN controller to cut its own connectivity to the remote node by a configuration impacting its own management connection into the node. The ACP itself should have no dedicated configuration other than aforementioned enablement of the ACP on brownfield ACP nodes. This leaves configuration that cannot distinguish between ACP and Data-Plane as sources of configuration mistakes as these commands will impact the ACP even though they should only impact the Data-Plane.

The one ubiquitous type of commands that do this on many type of routers are interface "down" commands/configurations. When such a command is applied to the interface through which the ACP provides access for remote management it would cut the remote management connection through the ACP because, as outlined above, the "down" commands typically impact the physical layer too and not only the Data-Plane services.

To provide ACP/ANI resilience against such operator misconfiguration, this document recommends to separate the "down" state of interfaces into an "admin down" state where the physical layer is kept running and ACP/ANI can use the interface and a "physical down" state. Any existing "down" configurations would map to "admin down". In "admin down", any existing L2/L3 services of the Data-Plane should see no difference to "physical down" state. To ensure that no Data-Plane packets could be sent/received, packet filtering could be established automatically as described above in Section 9.3.1.

An example of non-ACP but ANI traffic that should be permitted to pass even in "admin-down" state is BRSKI enrollment traffic between BRSKI pledge and a BRSKI proxy.

As necessary (see discussion below) new configuration options could be introduced to issue "physical down". The options should be provided with additional checks to minimize the risk of issuing them in a way that breaks the ACP without automatic restoration. For example, they could be denied to be issued from a control connection (NETCONF/SSH) that goes across the interface itself ("do not disconnect yourself"). Or they could be performed only temporary and only be made permanent with additional later reconfirmation.

In the following sub-sections important aspects to the introduction of "admin down" state are discussed.

#### 9.3.2.1. Security

Interfaces are physically brought down (or left in default down state) as a form of security. "Admin down" state as described above provides also a high level of security because it only permits ACP/ ANI operations which are both well secured. Ultimately, it is subject to security review for the deployment whether "admin down" is a feasible replacement for "physical down".

The need to trust the security of ACP/ANI operations needs to be weighed against the operational benefits of permitting this: Consider the typical example of a CPE (customer premises equipment) with no on-site network expert. User ports are in physical down state unless explicitly configured not to be. In a misconfiguration situation, the uplink connection is incorrectly plugged into such as user port. The device is disconnected from the network and therefore no diagnostics from the network side is possible anymore. Alternatively, all ports default to "admin down". The ACP (but not the Data-Plane) would still automatically form. Diagnostics from the network side is possible and operator reaction could include to either make this port the operational uplink port or to instruct re-cabling. Security wise, only ACP/ANI could be attacked, all other functions are filtered on interfaces in "admin down" state.

#### 9.3.2.2. Fast state propagation and Diagnostics

"Physical down" state propagates on many interface types (e.g., Ethernet) to the other side. This can trigger fast L2/L3 protocol reaction on the other side and "admin down" would not have the same (fast) result.

Bringing interfaces to "physical down" state is to the best of our knowledge always a result of operator action, but today, never the result of autonomic L2/L3 services running on the nodes. Therefore, one option is to change the operator action to not rely on link-state propagation anymore. This may not be possible when both sides are under different operator control, but in that case it is unlikely that the ACP is running across the link and actually putting the interface into "physical down" state may still be a good option.

Ideally, fast physical state propagation is replaced by fast software driven state propagation. For example, a DULL GRASP "admin-state" objective could be used to auto configure a Bidirectional Forwarding Protocol (BFD, [RFC5880]) session between the two sides of the link that would be used to propagate the "up" vs. admin down state.

Triggering physical down state may also be used as a mean of diagnosing cabling in the absence of easier methods. It is more complex than automated neighbor diagnostics because it requires coordinated remote access to both (likely) sides of a link to determine whether up/down toggling will cause the same reaction on the remote side.

See Section 9.1 for a discussion about how LLDP and/or diagnostics via GRASP could be used to provide neighbor diagnostics, and therefore hopefully eliminating the need for "physical down" for neighbor diagnostics - as long as both neighbors support ACP/ANI.

#### 9.3.2.3. Low Level Link Diagnostics

"Physical down" is performed to diagnose low-level interface behavior when higher layer services (e.g., IPv6) are not working. Especially Ethernet links are subject to a wide variety of possible wrong configuration/cablings if they do not support automatic selection of variable parameters such as speed (10/100/1000 Mbps), crossover (Auto-MDIX) and connector (fiber, copper - when interfaces have multiple but can only enable one at a time). The need for low level link diagnostic can therefore be minimized by using fully auto configuring links.

In addition to "Physical down", low level diagnostics of Ethernet or other interfaces also involve the creation of other states on interfaces, such as physical Loopback (internal and/or external) or bringing down all packet transmissions for reflection/cable-length measurements. Any of these options would disrupt ACP as well.

In cases where such low-level diagnostics of an operational link is desired but where the link could be a single point of failure for the ACP, ASA on both nodes of the link could perform a negotiated diagnostic that automatically terminates in a predetermined manner without dependence on external input ensuring the link will become operational again.

#### 9.3.2.4. Power Consumption Issues

Power consumption of "physical down" interfaces, may be significantly lower than those in "admin down" state, for example on long-range fiber interfaces. Bringing up interfaces, for example to probe reachability, may also consume additional power. This can make these type of interfaces inappropriate to operate purely for the ACP when they are not currently needed for the Data-Plane.

### 9.3.3. Interface level ACP/ANI enable

The interface level configuration option "ACP enable" enables ACP operations on an interface, starting with ACP neighbor discovery via DULL GRAP. The interface level configuration option "ANI enable" on nodes supporting BRSKI and ACP starts with BRSKI pledge operations when there is no domain certificate on the node. On ACP/BRSKI nodes, "ACP enable" may not need to be supported, but only "ANI enable". Unless overridden by global configuration options (see later), "ACP/ ANI enable" will result in "down" state on an interface to behave as "admin down".

### 9.3.4. Which interfaces to auto-enable?

(Section 6.4) requires that "ACP enable" is automatically set on native interfaces, but not on non-native interfaces (reminder: a native interface is one that exists without operator configuration action such as physical interfaces in physical devices).

Ideally, ACP enable is set automatically on all interfaces that provide access to additional connectivity that allows to reach more nodes of the ACP domain. The best set of interfaces necessary to achieve this is not possible to determine automatically. Native interfaces are the best automatic approximation.

Consider an ACP domain of ACP nodes transitively connected via native interfaces. A Data-Plane tunnel between two of these nodes that are non-adjacent is created and "ACP enable" is set for that tunnel. ACP RPL sees this tunnel as just as a single hop. Routes in the ACP would use this hop as an attractive path element to connect regions adjacent to the tunnel nodes. In result, the actual hop-by-hop paths used by traffic in the ACP can become worse. In addition, correct forwarding in the ACP now depends on correct Data-Plane forwarding config including QoS, filtering and other security on the Data-Plane path across which this tunnel runs. This is the main issue why "ACP/ANI enable" should not be set automatically on non-native interfaces.

If the tunnel would connect two previously disjoint ACP regions, then it likely would be useful for the ACP. A Data-Plane tunnel could also run across nodes without ACP and provide additional connectivity for an already connected ACP network. The benefit of this additional ACP redundancy has to be weighed against the problems of relying on the Data-Plane. If a tunnel connects two separate ACP regions: how many tunnels should be created to connect these ACP regions reliably enough? Between which nodes? These are all standard tunneled network design questions not specific to the ACP, and there are no generic fully automated answers.

Instead of automatically setting "ACP enable" on these type of interfaces, the decision needs to be based on the use purpose of the non-native interface and "ACP enable" needs to be set in conjunction with the mechanism through which the non-native interface is created/ configured.

In addition to explicit setting of "ACP/ANI enable", non-native interfaces also need to support configuration of the ACP RPL cost of the link - to avoid the problems of attracting too much traffic to the link as described above.

Even native interfaces may not be able to automatically perform BRSKI or ACP because they may require additional operator input to become operational. Example include DSL interfaces requiring PPPoE credentials or mobile interfaces requiring credentials from a SIM card. Whatever mechanism is used to provide the necessary config to the device to enable the interface can also be expanded to decide on whether or not to set "ACP/ANI enable".

The goal of automatically setting "ACP/ANI enable" on interfaces (native or not) is to eliminate unnecessary "touches" to the node to make its operation as much as possible "zero-touch" with respect to ACP/ANI. If there are "unavoidable touches" such a creating/ configuring a non-native interface or provisioning credentials for a native interface, then "ACP/ANI enable" should be added as an option to that "touch". If a wrong "touch" is easily fixed (not creating another high-cost touch), then the default should be not to enable ANI/ACP, and if it is potentially expensive or slow to fix (e.g., parameters on SIM card shipped to remote location), then the default should be to enable ACP/ANI.

#### 9.3.5. Node Level ACP/ANI enable

A node level command "ACP/ANI enable [up-if-only]" enables ACP or ANI on the node (ANI = ACP + BRSKI). Without this command set, any interface level "ACP/ANI enable" is ignored. Once set, ACP/ANI will operate an interface where "ACP/ANI enable" is set. Setting of interface level "ACP/ANI enable" is either automatic (default) or explicit through operator action as described in the previous section.

If the option "up-if-only" is selected, the behavior of "down" interfaces is unchanged, and ACP/ANI will only operate on interfaces where "ACP/ANI enable" is set and that are "up". When it is not set, then "down" state of interfaces with "ACP/ANI enable" is modified to behave as "admin down".

##### 9.3.5.1. Brownfield nodes

A "brownfield" node is one that already has a configured Data-Plane.

Executing global "ACP/ANI enable [up-if-only]" on each node is the only command necessary to create an ACP across a network of brownfield nodes once all the nodes have a domain certificate. When BRSKI is used ("ANI enable"), provisioning of the certificates only requires set-up of a single BRSKI registrar node which could also implement a CA for the network. This is the simplest way to introduce ACP/ANI into existing (== brownfield) networks.

The need to explicitly enable ACP/ANI is especially important in brownfield nodes because otherwise software updates may introduce support for ACP/ANI: Automatic enablement of ACP/ANI in networks where the operator does not only not want ACP/ANI but where the operator likely never even heard of it could be quite irritating to the operator. Especially when "down" behavior is changed to "admin down".

Automatically setting "ANI enable" on brownfield nodes where the operator is unaware of BRSKI and MASA operations could also be an unlikely but then critical security issue. If an attacker could impersonate the operator and register as the operator at the MASA or otherwise get hold of vouchers and can get enough physical access to the network so pledges would register to an attacking registrar, then the attacker could gain access to the ACP, and through the ACP gain access to the Data-Plane.

In networks where the operator explicitly wants to enable the ANI this could not happen, because the operator would create a BRSKI registrar that would discover attack attempts, and the operator would be setting up his registrar with the MASA. Nodes requiring "ownership vouchers" would not be subject to that attack. See [I-D.ietf-anima-bootstrapping-keyinfra] for more details. Note that a global "ACP enable" alone is not subject to these type of attacks, because it always depends on some other mechanism first to provision domain certificates into the device.

#### 9.3.5.2. Greenfield nodes

An ACP "greenfield" node is one that does not have any prior configuration and that can be bootstrapped into the ACP across the network. To support greenfield nodes, ACP as described in this document needs to be combined with a bootstrap protocol/mechanism that will enroll the node with the ACP keying material - ACP certificate and TA. For ANI nodes, this protocol/mechanism is BRSKI.

When such a node is powered on and determines it is in greenfield condition, it enables the bootstrap protocol(s)/mechanism(s), and once the ACP keying material is enrolled, greenfield state ends and the ACP is started. When BRSKI is used, the node's state reflects this by setting "ANI enable" upon determination of greenfield state at power on.

ACP greenfield nodes that in the absence of ACP would have their interfaces in "down" state SHOULD set all native interfaces into "admin down" state and only permit Data-Plane traffic required for the bootstrap protocol/mechanisms.

ACP greenfield state ends either through successful enrolment of ACP keying material (certificate, TA) or detection of a permitted termination of ACP greenfield operations.

ACP nodes supporting greenfield operations MAY want to provide backward compatibility with other forms of configuration/ provisioning, especially when only a subset of nodes are expected to be deployed with ACP. Such an ACP node SHOULD observe attempts to provision/configure the node via interfaces/methods that traditionally indicate physical possession of the node, such as a serial or USB console port or a USB memory stick with a bootstrap configuration. When such an operation is observed before enrollment of the ACP keying material has completed, the node SHOULD put itself into the state the node would have been in, if ACP/ANI was disabled at boot (terminate ACP greenfield operations).

When an ACP greenfield node enables multiple automated ACP or non-ACP enrollment/bootstrap protocols/mechanisms in parallel, care must be taken not to terminate any protocol/mechanism before another one has succeeded to enroll ACP keying material or has progressed to a point where it is permitted to be a termination reason for ACP greenfield operations.

Highly secure ACP greenfield nodes may not permit any reason to terminate ACP greenfield operations, including physical access.

Nodes that claim to support ANI greenfield operations SHOULD NOT enable in parallel to BRSKI any enrollment/bootstrap protocol/ mechanism that allows Trust On First Use (TOFU, [RFC7435]) over interfaces other than those traditionally indicating physical possession of the node. Protocols/mechanisms with published default username/password authentication are considered to suffer from TOFU. Securing the bootstrap protocol/mechanism by requiring a voucher ([RFC8366]) can be used to avoid TOFU.

In summary, the goal of ACP greenfield support is to allow remote automated enrollment of ACP keying materials, and therefore automated bootstrap into the ACP and to prohibit TOFU during bootstrap with the likely exception (for backward compatibility) of bootstrapping via interfaces traditionally indicating physical possession of the node.

#### 9.3.6. Undoing ANI/ACP enable

Disabling ANI/ACP by undoing "ACP/ANI enable" is a risk for the reliable operations of the ACP if it can be executed by mistake or unauthorized. This behavior could be influenced through some additional (future) property in the certificate (e.g., in the acp-node-name extension field): In an ANI deployment intended for convenience, disabling it could be allowed without further constraints. In an ANI deployment considered to be critical more checks would be required. One very controlled option would be to not permit these commands unless the domain certificate has been revoked or is denied renewal. Configuring this option would be a parameter on the BRSKI registrar(s). As long as the node did not receive a domain certificate, undoing "ANI/ACP enable" should not have any additional constraints.

#### 9.3.7. Summary

Node-wide "ACP/ANI enable [up-if-only]" commands enable the operation of ACP/ANI. This is only auto-enabled on ANI greenfield devices, otherwise it must be configured explicitly.

If the option "up-if-only" is not selected, interfaces enabled for ACP/ANI interpret "down" state as "admin down" and not "physical down". In "admin-down" all non-ACP/ANI packets are filtered, but the physical layer is kept running to permit ACP/ANI to operate.

(New) commands that result in physical interruption ("physical down", "loopback") of ACP/ANI enabled interfaces should be built to protect continuance or reestablishment of ACP as much as possible.

Interface level "ACP/ANI enable" control per-interface operations. It is enabled by default on native interfaces and has to be configured explicitly on other interfaces.

Disabling "ACP/ANI enable" global and per-interface should have additional checks to minimize undesired breakage of ACP. The degree of control could be a domain wide parameter in the domain certificates.

### 9.4. Partial or Incremental adoption

The ACP Zone Addressing Sub-Scheme (see Section 6.11.3) allows incremental adoption of the ACP in a network where ACP can be deployed on edge areas, but not across the core that is connecting those edges.

In such a setup, each edge network, such as a branch or campus of an enterprise network has a disjoined ACP to which one or more unique Zone-IDs are assigned: ACP nodes registered for a specific ACP zone have to receive ACP Zone Addressing Sub-scheme addresses, for example by virtue of configuring for each such zone one or more ACP Registrars with that Zone-ID. All the Registrars for these ACP Zones need to get ACP certificates from CAs relying on a common set of TA and of course the same ACP domain name.

These ACP zones can first be brought up as separate networks without any connection between them and/or they can be connected across a non-ACP enabled core network through various non-autonomic operational practices. For example, each separate ACP Zone can have an edge node that is a layer 3 VPN PE (MPLS or IPv6 layer 3 VPN), where a complete non-autonomic ACP-Core VPN is created by using the ACP VRFs and exchanging the routes from those ACP VRFs across the VPNs non-autonomic routing protocol(s).

While such a setup is possible with any ACP addressing sub-scheme, the ACP-Zone Addressing sub-scheme makes it easy to configure and scalable for any VPN routing protocols because every ACP zone would only need to indicate one or more /64 ACP Zone Addressing prefix routes into the ACP-Core VPN as opposed to routes for every individual ACP node as required in the other ACP addressing schemes.

Note that the non-autonomous ACP-Core VPN would require additional extensions to propagate GRASP messages when GRASP discovery is desired across the zones.

For example, one could set up on each Zone edge router a remote ACP tunnel to a GRASP hub. The GRASP hub could be implemented at the application level and could run in the NOC of the network. It would serve to propagate GRASP announcements between ACP Zones and/or generate GRASP announcements for NOC services.

Such a partial deployment may prove to be sufficient or could evolve to become more autonomous through future standardized or non-standardized enhancements, for example by allowing GRASP messages to be propagated across the layer 3 VPN, leveraging for example L3VPN Multicast support.

Finally, these partial deployments can be merged into a single contiguous complete autonomous ACP (given appropriate ACP support across the core) without changes in the crypto material, because the node's ACP certificates are from a single ACP.

## 9.5. Configuration and the ACP (summary)

There is no desirable configuration for the ACP. Instead, all parameters that need to be configured in support of the ACP are limitations of the solution, but they are only needed in cases where not all components are made autonomic. Wherever this is necessary, it relies on pre-existing mechanisms for configuration such as CLI or YANG ([RFC7950]) data models.

The most important examples of such configuration include:

* When ACP nodes do not support an autonomic way to receive an ACP certificate, for example BRSKI, then such certificate needs to be configured via some pre-existing mechanisms outside the scope of this specification. Today, router have typically a variety of mechanisms to do this.
* Certificate maintenance requires PKI functions. Discovery of these functions across the ACP is automated (see Section 6.2.5), but their configuration is not.

* When non-ACP capable nodes such as pre-existing NMS need to be physically connected to the ACP, the ACP node to which they attach needs to be configured with ACP-connect according to Section 8.1. It is also possible to use that single physical connection to connect both to ACP and the Data-Plane of the network as explained in Section 8.1.4.
* When devices are not autonomically bootstrapped, explicit configuration to enable the ACP needs to be applied. See Section 9.3.
* When the ACP needs to be extended across interfaces other than L2, the ACP as defined in this document cannot autodiscover candidate neighbors automatically. Remote neighbors need to be configured, see Section 8.2.

Once the ACP is operating, any further configuration for the Data-Plane can be configured more reliably across the ACP itself because the ACP provides addressing and connectivity (routing) independent of the Data-Plane itself. For this, the configuration methods simply need to also allow to operate across the ACP VRF - NETCONF, SSH or any other method.

The ACP also provides additional security through its hop-by-hop encryption for any such configuration operations: Some legacy configuration methods (SNMP, TFTP, HTTP) may not use end-to-end encryption, and most of the end-to-end secured configuration methods still allow for easy passive observation along the path about configuration taking place (transport flows, port numbers, IP addresses).

The ACP can and should equally be used as the transport to configure any of the aforementioned non-autonomic components of the ACP, but in that case, the same caution needs to be exercised as with Data-Plane configuration without ACP: Misconfiguration may cause the configuring entity to be disconnected from the node it configures - for example when incorrectly unconfiguring a remote ACP neighbor through which the configured ACP node is reached.

## 10. Summary: Benefits (Informative)

### 10.1. Self-Healing Properties

The ACP is self-healing:

* New neighbors will automatically join the ACP after successful validation and will become reachable using their unique ULA address across the ACP.

* When any changes happen in the topology, the routing protocol used in the ACP will automatically adapt to the changes and will continue to provide reachability to all nodes.
* The ACP tracks the validity of peer certificates and tears down ACP secure channels when a peer certificate has expired. When short-lived certificates with lifetimes in the order of OCSP/CRL refresh times are used, then this allows for removal of invalid peers (whose certificate was not renewed) at similar speeds as when using OCSP/CRL. The same benefit can be achieved when using CRL/OCSP, periodically refreshing the revocation information and also tearing down ACP secure channels when the peer's (long-lived) certificate is revoked. There is no requirement against ACP implementations to require this enhancement though to keep the mandatory implementations simpler.

The ACP can also sustain network partitions and mergers. Practically all ACP operations are link local, where a network partition has no impact. Nodes authenticate each other using the domain certificates to establish the ACP locally. Addressing inside the ACP remains unchanged, and the routing protocol inside both parts of the ACP will lead to two working (although partitioned) ACPs.

There are few central dependencies: A CRL may not be available during a network partition; a suitable policy to not immediately disconnect neighbors when no CRL is available can address this issue. Also, an ACP Registrar or Certification Authority might not be available during a partition. This may delay renewal of certificates that are to expire in the future, and it may prevent the enrollment of new nodes during the partition.

Highly resilient ACP designs can be built by using ACP Registrars with embedded sub-CA, as outlined in Section 9.2.4. As long as a partition is left with one or more of such ACP Registrars, it can continue to enroll new candidate ACP nodes as long as the ACP Registrar's sub-CA certificate does not expire. Because the ACP addressing relies on unique Registrar-IDs, a later re-merge of partitions will also not cause problems with ACP addresses assigned during partitioning.

After a network partition, a re-merge will just establish the previous status, certificates can be renewed, the CRL is available, and new nodes can be enrolled everywhere. Since all nodes use the same TA, a re-merge will be smooth.

Merging two networks with different TA requires the ACP nodes to trust the union of TA. As long as the routing-subdomain hashes are different, the addressing will not overlap. Accidentally, overlaps will only happen in the unlikely event of a 40-bit hash collision in SHA256 (see Section 6.11). Note that the complete mechanisms to merge networks is out of scope of this specification.

It is also highly desirable for implementation of the ACP to be able to run it over interfaces that are administratively down. If this is not feasible, then it might instead be possible to request explicit operator override upon administrative actions that would administratively bring down an interface across which the ACP is running. Especially if bringing down the ACP is known to disconnect the operator from the node. For example, any such down administrative action could perform a dependency check to see if the transport connection across which this action is performed is affected by the down action (with default RPL routing used, packet forwarding will be symmetric, so this is actually possible to check).

## 10.2. Self-Protection Properties

### 10.2.1. From the outside

As explained in Section 6, the ACP is based on secure channels built between nodes that have mutually authenticated each other with their domain certificates. The channels themselves are protected using standard encryption technologies such as DTLS or IPsec which provide additional authentication during channel establishment, data integrity and data confidentiality protection of data inside the ACP and in addition, provide replay protection.

Attacker will not be able to join the ACP unless they have a valid ACP certificate. On-path attackers without a valid ACP certificate cannot inject packets into the ACP due to ACP secure channels. They can also not decrypt ACP traffic except if they can crack the encryption. They can attempt behavioral traffic analysis on the encrypted ACP traffic.

The degree to which compromised ACP nodes can impact the ACP depends on the implementation of the ACP nodes and their impairment. When an attacker has only gained administrative privileges to configure ACP nodes remotely, the attacker can disrupt the ACP only through one of the few configuration options to disable it, see Section 9.3, or by configuring of non-autonomic ACP options if those are supported on the impaired ACP nodes, see Section 8. Injecting or extracting traffic into/from an impaired ACP node is only possible when an impaired ACP node supports ACP connect (see Section 8.1) and the attacker can control traffic into/from one of the ACP nodes interfaces, such as by having physical access to the ACP node.

The ACP also serves as protection (through authentication and encryption) for protocols relevant to OAM that may not have secured protocol stack options or where implementation or deployment of those options fail on some vendor/product/customer limitations. This includes protocols such as SNMP ([RFC3411]), NTP ([RFC5905]), PTP ([IEEE-1588-2008]), DNS ([RFC3596]), DHCPv6 ([RFC3315]), syslog ([RFC3164]), RADIUS ([RFC2865]), Diameter ([RFC6733]), TACACS ([RFC1492]), IPFIX ([RFC7011]), Netflow ([RFC3954]) - just to name a few. Not all of these protocol references are necessarily the latest version of protocols but versions that are still widely deployed.

Protection via the ACP secure hop-by-hop channels for these protocols is meant to be only a stopgap though: The ultimate goal is for these and other protocols to use end-to-end encryption utilizing the domain certificate and rely on the ACP secure channels primarily for zero-touch reliable connectivity, but not primarily for security.

The remaining attack vector would be to attack the underlying ACP protocols themselves, either via directed attacks or by denial-of-service attacks. However, as the ACP is built using link-local IPv6 addresses, remote attacks from the Data-Plane are impossible as long as the Data-Plane has no facilities to remotely send IPv6 link-local packets. The only exceptions are ACP connected interfaces which require higher physical protection. The ULA addresses are only reachable inside the ACP context, therefore, unreachable from the Data-Plane. Also, the ACP protocols should be implemented to be attack resistant and not consume unnecessary resources even while under attack.

#### 10.2.2. From the inside

The security model of the ACP is based on trusting all members of the group of nodes that receive an ACP certificate for the same domain. Attacks from the inside by a compromised group member are therefore the biggest challenge.

Group members must be protected against attackers so that there is no easy way to compromise them, or use them as a proxy for attacking other devices across the ACP. For example, management plane functions (transport ports) should only be reachable from the ACP but not the Data-Plane. Especially for those management plane functions that have no good protection by themselves because they do not have secure end-to-end transport and to whom ACP not only provides automatic reliable connectivity but also protection against attacks. Protection across all potential attack vectors is typically easier to do in devices whose software is designed from the ground up with ACP in mind than with legacy software based systems where the ACP is added on as another feature.

As explained above, traffic across the ACP should still be end-to-end encrypted whenever possible. This includes traffic such as GRASP, EST and BRSKI inside the ACP. This minimizes man in the middle attacks by compromised ACP group members. Such attackers cannot eavesdrop or modify communications, they can just filter them (which is unavoidable by any means).

See Appendix A.9.8 for further considerations how to avoid and deal with compromised nodes.

## 10.3. The Administrator View

An ACP is self-forming, self-managing and self-protecting, therefore has minimal dependencies on the administrator of the network. Specifically, since it is (intended to be) independent of configuration, there is only limited scope for configuration errors on the ACP itself. The administrator may have the option to enable or disable the entire approach, but detailed configuration is not possible. This means that the ACP must not be reflected in the running configuration of nodes, except a possible on/off switch (and even that is undesirable).

While configuration (except for Section 8 and Section 9.2) is not possible, an administrator must have full visibility of the ACP and all its parameters, to be able to do trouble-shooting. Therefore, an ACP must support all show and debug options, as for any other network function. Specifically, a network management system or controller must be able to discover the ACP, and monitor its health. This visibility of ACP operations must clearly be separated from visibility of Data-Plane so automated systems will never have to deal with ACP aspects unless they explicitly desire to do so.

Since an ACP is self-protecting, a node not supporting the ACP, or without a valid domain certificate cannot connect to it. This means that by default a traditional controller or network management system cannot connect to an ACP. See Section 8.1.1 for more details on how to connect an NMS host into the ACP.

## 11. Security Considerations

A set of ACP nodes with ACP certificates for the same ACP domain and with ACP functionality enabled is automatically "self-building": The ACP is automatically established between neighboring ACP nodes. It is also "self-protecting": The ACP secure channels are authenticated and encrypted. No configuration is required for this.

The self-protecting property does not include workarounds for non-autonomic components as explained in Section 8. See Section 10.2 for details of how the ACP protects itself against attacks from the outside and to a more limited degree from the inside as well.

However, the security of the ACP depends on a number of other factors:

* The usage of domain certificates depends on a valid supporting PKI infrastructure. If the chain of trust of this PKI infrastructure is compromised, the security of the ACP is also compromised. This is typically under the control of the network administrator.
* ACP nodes receive their certificates from ACP registrars. These ACP registrars are security critical dependencies of the ACP: Procedures and protocols for ACP registrars are outside the scope of this specification as explained in Section 6.11.7.1, only requirements against the resulting ACP certificates are specified.
* Every ACP registrar (for enrollment of ACP certificates) and ACP EST server (for renewal of ACP certificates) is a security critical entity and its protocols are security critical protocols. Both need to be hardened against attacks, similar to a CA and its protocols. A malicious registrar can enroll malicious nodes to an ACP network (if the CA delegates this policy to the registrar) or break ACP routing for example by assigning duplicate ACP address assignment to ACP nodes via their ACP certificates.
* ACP nodes that are ANI nodes rely on BRSKI as the protocol for ACP registrars. For ANI type ACP nodes, the security considerations of BRSKI apply. It enables automated, secure enrollment of ACP certificates.
* BRSKI and potentially other ACP registrar protocol options require that nodes have an (X.509v3 based) IDevID. IDevIDs are an option for ACP registrars to securely identify candidate ACP nodes that should be enrolled into an ACP domain.

* For IDevIDs to securely identify the node to which it IDevID is assigned, the node needs to (1) utilize hardware support such as a Trusted Platform Module (TPM) to protect against extraction/cloning of the private key of the IDevID and (2) a hardware/software infrastructure to prohibit execution of non-authenticated software to protect against malicious use of the IDevID.
* Like the IDevID, the ACP certificate should equally be protected from extraction or other abuse by the same ACP node infrastructure. This infrastructure for IDevID and ACP certificate is beneficial independent of the ACP registrar protocol used (BRSKI or other).
* Renewal of ACP certificates requires support for EST, therefore the security considerations of [RFC7030] related to certificate renewal/rekeying and TP renewal apply to the ACP. EST security considerations when using other than mutual certificate authentication do not apply nor do considerations for initial enrollment via EST apply, except for ANI type ACP nodes because BRSKI leverages EST.
* A malicious ACP node could declare itself to be an EST server via GRASP across the ACP if malicious software could be executed on it. CA should therefore authenticate only known trustworthy EST servers, such as nodes with hardware protections against malicious software. When Registrars use their ACP certificate to authenticate towards a CA, the id-kp-cmcRA [RFC6402] extended key usage attribute allows the CA to determine that the ACP node was permitted during enrollment to act as an ACP registrar. Without the ability to talk to the CA, a malicious EST server can still attract ACP nodes attempting to renew their keying material, but they will fail to perform successful renewal of a valid ACP certificate. The ACP node attempting to use the malicious EST server can then continue to use a different EST server, and log a failure against a malicious EST server.
* Malicious on-path ACP nodes may filter valid EST server announcements across the ACP, but such malicious ACP nodes could equally filter any ACP traffic such as the EST traffic itself. Either attack requires the ability to execute malicious software on an impaired ACP node though.
* In the absence of malicious software injection, an attacker that can misconfigure an ACP node which is supporting EST server functionality could attempt to configure a malicious CA. This would not result in the ability to successfully renew ACP certificates, but it could result in DoS attacks by becoming an EST server and making ACP nodes attempting their ACP certificate renewal via this impaired ACP node. This problem can be avoided when the EST server implementation can verify that the CA configured is indeed providing renewal for certificates of the node's ACP. The ability to do so depends on the EST-Server to CA protocol, which is outside the scope of this document.

In summary, attacks against the PKI/certificate dependencies of the ACP can be minimized by a variety of hardware/software components including options such as TPM for IDevID/ACP-certificate, prohibitions against execution of non-trusted software and design aspects of the EST Server functionality for the ACP to eliminate configuration level impairment.

Because ACP peers select one out of potentially more than one mutually supported ACP secure channel protocols via the approach described in Section 6.6, ACP secure channel setup is subject to downgrade attacks by MITM attackers. This can be discovered after such an attack by additional mechanisms described in Appendix A.9.9. Alternatively, more advanced channel selection mechanisms can be devised. [RFC-Editor: Please remove the following sentence]. See [ACPDRAFT] Appendix B.1. Both options are out of scope of this document.

The security model of the ACP as defined in this document is tailored for use with private PKI. The TA of a private PKI provide the security against maliciously created ACP certificates to give access to an ACP. Such attacks can create fake ACP certificates with correct looking AcpNodeNames, but those certificates would not pass the certificate path validation of the ACP domain membership check (see Section 6.2.3, point 2).

[RFC-Editor: please remove the following paragraph].

Using public CA is out of scope of this document. See [ACPDRAFT], Appendix B.3 for further considerations.

There is no prevention of source-address spoofing inside the ACP. This implies that if an attacker gains access to the ACP, it can spoof all addresses inside the ACP and fake messages from any other node. New protocol/services run across the ACP should therefore use end-to-end authentication inside the ACP. This is already done by GRASP as specified in this document.

The ACP is designed to enable automation of current network management and future autonomic peer-to-peer/distributed network automation. Any ACP member can send ACP IPv6 packet to other ACP members and announce via ACP GRASP services to all ACP members without dependency against centralized components.

The ACP relies on peer-to-peer authentication and authorization using ACP certificates. This security model is necessary to enable the autonomic ad-hoc any-to-any connectivity between ACP nodes. It provides infrastructure protection through hop by hop authentication and encryption - without relying on third parties. For any services where this complete autonomic peer-to-peer group security model is appropriate, the ACP certificate can also be used unchanged. For example, for any type of Data-Plane routing protocol security.

This ACP security model is designed primarily to protect against attack from the outside, but not against attacks from the inside. To protect against spoofing attacks from compromised on-path ACP nodes, end-to-end encryption inside the ACP is used by new ACP signaling: GRASP across the ACP using TLS. The same is expected from any non-legacy services/protocols using the ACP. Because no group-keys are used, there is no risk for impacted nodes to access end-to-end encrypted traffic from other ACP nodes.

Attacks from impacted ACP nodes against the ACP are more difficult than against the Data-Plane because of the autoconfiguration of the ACP and the absence of configuration options that could be abused that allow to change/break ACP behavior. This is excluding configuration for workaround in support of non-autonomic components.

Mitigation against compromised ACP members is possible through standard automated certificate management mechanisms including revocation and non-renewal of short-lived certificates. In this version of the specification, there are no further optimization of these mechanisms defined for the ACP (but see Appendix A.9.8).

Higher layer service built using ACP certificates should not solely rely on undifferentiated group security when another model is more appropriate/more secure. For example, central network configuration relies on a security model where only few especially trusted nodes are allowed to configure the Data-Plane of network nodes (CLI, NETCONF). This can be done through ACP certificates by differentiating them and introduce roles. See Appendix A.9.5.

Operators and provisioning software developers need to be aware of how the provisioning/configuration of network devices impacts the ability of the operator / provisioning software to remotely access the network nodes. By using the ACP, most of the issues of configuration/provisioning caused loss of connectivity for remote provisioning/configuration will be eliminated, see Section 6. Only few exceptions such as explicit physical interface down configuration will be left Section 9.3.2.

Many details of ACP are designed with security in mind and discussed elsewhere in the document:

IPv6 addresses used by nodes in the ACP are covered as part of the node's domain certificate as described in Section 6.2.2. This allows even verification of ownership of a peer's IPv6 address when using a connection authenticated with the domain certificate.

The ACP acts as a security (and transport) substrate for GRASP inside the ACP such that GRASP is not only protected by attacks from the outside, but also by attacks from compromised inside attackers - by relying not only on hop-by-hop security of ACP secure channels, but adding end-to-end security for those GRASP messages. See Section 6.9.2.

ACP provides for secure, resilient zero-touch discovery of EST servers for certificate renewal. See Section 6.2.5.

ACP provides extensible, auto-configuring hop-by-hop protection of the ACP infrastructure via the negotiation of hop-by-hop secure channel protocols. See Section 6.6.

The ACP is designed to minimize attacks from the outside by minimizing its dependency against any non-ACP (Data-Plane) operations/configuration on a node. See also Section 6.13.2.

In combination with BRSKI, ACP enables a resilient, fully zero-touch network solution for short-lived certificates that can be renewed or re-enrolled even after unintentional expiry (e.g., because of interrupted connectivity). See Appendix A.2.

Because ACP secure channels can be long lived, but certificates used may be short lived, secure channels, for example built via IPsec need to be terminated when peer certificates expire. See Section 6.8.5.

Section 7.2 describes how to implement a routed ACP topology operating on what effectively is a large bridge-domain when using L3/L2 routers that operate at L2 in the Data-Plane. In this case, the ACP is subject to much higher likelihood of attacks by other nodes "stealing" L2 addresses than in the actual routed case. Especially when the bridged network includes non-trusted devices such as hosts. This is a generic issue in L2 LANs. L2/L3 devices often already have some form of "port security" to prohibit this. They rely on NDP or DHCP learning of which port/MAC-address and IPv6 address belong together and block MAC/IPv6 source addresses from wrong ports. This type of function needs to be enabled to prohibit DoS attacks and specifically to protect the ACP. Likewise the GRASP DULL instance needs to ensure that the IPv6 address in the locator-option matches the source IPv6 address of the DULL GRASP packet.

## 12. IANA Considerations

This document defines the "Autonomic Control Plane".

For the ANIMA-ACP-2020 ASN.1 module, IANA is asked to register value IANA1 for "id-mod-anima-acpnodename-2020" in the "SMI Security for PKIX Module Identifier" (1.3.6.1.5.5.7.0) registry.

For the otherName / AcpNodeName, IANA is asked to register a value for IANA2 for id-on-AcpNodeName in the "SMI Security for PKIX Other Name Forms" (1.3.6.1.5.5.7.8) registry.

The IANA is requested to register the value "AN_ACP" (without quotes) to the GRASP Objectives Names Table in the GRASP Parameter Registry. The specification for this value is this document, Section 6.4.

The IANA is requested to register the value "SRV.est" (without quotes) to the GRASP Objectives Names Table in the GRASP Parameter Registry. The specification for this value is this document, Section 6.2.5.

Explanation: This document chooses the initially strange looking format "SRV.<service-name>" because these objective names would be in line with potential future simplification of the GRASP objective registry. Today, every name in the GRASP objective registry needs to be explicitly allocated with IANA. In the future, this type of objective names could be considered to be automatically registered in that registry for the same service for which a <service-nameh> is registered according to [RFC6335]. This explanation is solely informational and has no impact on the requested registration.

The IANA is requested to create an ACP Parameter Registry with currently one registry table - the "ACP Address Type" table.

"ACP Address Type" Table. The value in this table are numeric values 0...3 paired with a name (string). Future values MUST be assigned using the Standards Action policy defined by [RFC8126]. The following initial values are assigned by this document:

0: ACP Zone Addressing Sub-Scheme (ACP RFC Section 6.11.3)

1: ACP Vlong Addressing Sub-Scheme (ACP RFC Section 6.11.5) / ACP Manual Addressing Sub-Scheme (ACP RFC Section 6.11.4)

## 13. Acknowledgements

This work originated from an Autonomic Networking project at Cisco Systems, which started in early 2010. Many people contributed to this project and the idea of the Autonomic Control Plane, amongst which (in alphabetical order): Ignas Bagdonas, Parag Bhide, Balaji BL, Alex Clemm, Yves Hertoghs, Bruno Klauser, Max Pritikin, Michael Richardson, Ravi Kumar Vadapalli.

Special thanks to Brian Carpenter, Elwyn Davies, Joel Halpern and Sheng Jiang for their thorough reviews.

Many thanks Ben Kaduk, Roman Danyliv and Eric Rescorla for their thorough SEC AD reviews, Russ Housley and Erik Kline for their reviews and to Valery Smyslov, Tero Kivinen, Paul Wouters and Yoav Nir for review of IPsec and IKEv2 parameters and helping to understand those and other security protocol details better. Thanks for Carsten Borman for CBOR/CDDL help.

Further input, review or suggestions were received from: Rene Struik, Benoit Claise, William Atwood and Yongkang Zhang.

## 14. Contributors

For all things GRASP including validation code, ongoing document text support and technical input.

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

Email: brian.e.carpenter@gmail.com

For RPL contributions and all things BRSKI/bootstrap including validation code, ongoing document text support and technical input.

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca
URI: http://www.sandelman.ca/mcr/

For the RPL technology choices and text.

Pascal Thubert
Cisco Systems, Inc
Building D
45 Allee des Ormes - BP1200
06254 MOUGINS - Sophia Antipolis
France

Phone: +33 497 23 26 34
Email: pthubert@cisco.com

## 15. Change log [RFC-Editor: Please remove]

This document was developed on https://github.com/anima-wg/autonomic-control-plane/tree/master/draft-ietf-anima-autonomic-control-plane. That github repository also contains the document review/reply emails.

### 15.1. Summary of changes since entering IESG review

This text replaces the prior changelog with a summary to provide guidance for further IESG review.

Please see revision -21 for the individual changelogs of prior versions .

#### 15.1.1. Reviews (while in IESG review status) / status

This document entered IESG review with version -13. It has since seen the following reviews:

IESG: Original owner/Yes: Terry Manderson (INT).

IESG: No Objection: Deborah Brungard (RTG), Alissa Cooper (GEN), Warren Kumari (OPS), Mirja Kuehlewind (TSV), Alexey Melnikov (ART), Adam Roach (ART).

IESG: No Objection, not counted anymore as they have left IESG: Ben Campbell (ART), Spencer Dawkins (TSV).

IESG: Open DISCUSS hopefully resolved by this version: Eric Rescorla (SEC, left IESG), Benjamin Kaduk (SEC).

Other: Michael Richardson (WG), Brian Carpenter (WG), Pascal Thubert (WG), Frank Xialiang (WG), Elwyn Davies (GEN), Joel Halpern (RTGdir), Yongkang Zhang (WG), William Atwood (WG).

#### 15.1.2. BRSKI / ACP registrar related enhancements

Only after ACP entered IESG review did it become clear that the in-progress BRSKI document would not provide all the explanations needed for ACP registrars as expected earlier by ACP authors. Instead, BRSKI will only specify a subset of required ACP behavior related to certificate handling and registrar. There, it became clear that the ACP draft should specify generic ACP registrar behavior independent of BRSKI so ACP could be implemented with or without BRSKI and any manual/proprietary or future standardized BRSKI alternatives (for example via NETCONF) would understand the requirements for ACP registrars and its certificate handling.

This lead to additional text about ACP registrars in the ACP document:

1. Defined relationship ACP / ANI (ANI = ACP + BRSKI).

6.1.4 (new) Overview of TA required for ACP.

6.1.5.5 Added explanations/requirements for Re-enrollment.

6.10.7 Normative requirements for ACP registrars (BRSKI or not).

10.2 Operational expectations against ACP registrars (BRSKI or not).

#### 15.1.3. Normative enhancements since start of IESG review

In addition to above ACP registrar / BRSKI related enhancements there is a range of minor normative (also explanatory) enhancements since the start of IESG review:

6.1.1 Hex digits in ACP domain information field now upper-case (no specific reason except that both options are equally good, but capitalized ones are used in rfc5234).

6.1.5.3 Added explanations about CRLs.

6.1.5.6 Added explanations of behavior under failing certificates.

6.1.2 Allow ACP address '0' in ACP domain information field: presence of address indicates permission to build ACP secure channel to node, 0 indicates that the address of the node is assigned by (future) other means than certificate. Non-autonomic nodes have no address at all (that was in -13), and can only connect via ACP connect interfaces to ACP.

6.1.3 Distinction of real ACP nodes (with address) and those with domain certificate without address added as a new rule to ACP domain membership check.

6.6 Added throttling of secure-channel setup attempts.

6.11.1.14 Removed requirement to handle unknown destination ACP traffic in low-end nodes that would never be RPL roots.

6.12.5 Added recommendation to use IPv6 DAD.

6.1.1, 6.7.1.1, 6.7.2, 6.7.3, 6.8.2 Various refined additional certificate, secure channel protocol (IPsec/IKEv2 and DTLS) and ACP GRASP TLS protocol parameter requirements to ensure interoperating implementations (from SEC-AD review).

#### 15.1.4. Explanatory enhancements since start of IESG review

Beyond the functional enhancements from the previous two sections, the mayority of changes since -13 are additional explanations from review feedback, textual nits and restructuring - with no functional requirement additions/changes.

1.1 Added "applicability and scope" section with summarized explanations.

2.Added in-band vs. out-of-band management definitions.

6.1.2 (was 6.1.1) expanded explanations of reasoning for elements of the ACP domain information field.

6.1.3 refined explanations of ACP domain membership check and justifications for it.

6.5 Elaborated step-by-step secure channel setup.

6.10 Additional explanations for addressing modes, additional table of addressing formats (thanks MichaelR).

6.10.5 introduced 'F' bit position as a better visual representation in the Vlong address space.

6.11.1.1 extensive overhaul to improve readability of use of RPL (from IESG feedback of non-routing/RPL experts).

6.12.2 Added caution about unconfiguring Data-Plane IPv6 addresses and impact to ACP (limitation of current ACP design, and pointint to more details in 10.2).

10.4 New explanations / summary of configurations for ACP (aka: all config is undesirable and only required for integrating with non-autonomic components, primarily ACP-connect and Registrars).

11.  Textually enhanced / better structured security considerations section after IESG security review.

A. (new) Moved all explanations and discussions about futures from section 10 into this new appendix.  This text should not be removed because it captures a lot of repeated asked questions in WG and during reviews and from users, and also captures ideas for some likely important followup work.  But none of this is relevant to implementing (section 6) and operating (section 10) the ACP.

## 15.2.  draft-ietf-anima-autonomic-control-plane-30

-29 did pass all IESG DISCUSS.  This version cleans up reamining comments.

Planned to be removed section Appendix A.6 was moved into new Appendix B.1 to be amended by further A.2, A.3 containing text felt to be unfit for publication in RFC (see below).  Added reference to this last draft, and referencing those sections ([ACPDRAFT]).

Final discussion with responsible AD (Eric Vyncke): marked all references to [ACPDRAFT] as to be removed from RFC, as this would be too unconventional.  Likewise also [ACPDRAFT] reference itself. Added explanation to appendix B.

Comments from Erik Kline:

2.  Fine tuned ULA definition.

Comments Michael Richardson / Eric Vyncke.

6.2.4. / 11.  Removed text arguing ability how to use public CA (or not).  Replaced with reference to new [ACPDRAFT] section B.3 (not in RFC) that explains current state of understanding (unfinished).

B.3 New text detailling authors understanding of use of public CA (will not be in RFC).

Comments/proposals from Ben Kaduk:

Various: Replaced RFC4492 with RFC8422 which is superceeding it.

6.1 Text fix for hash strength 384 bits (from SHA384); Text fix for ec_point_format extension.

6.2.1 Text fixup. Removed requirements for ECDH support in certificate, instead merely explaining the dependencies required IF this is desired (educational).

6.2.5.4. Fine tuning 2 sentences.

6.3.2. (ACP domain memebership check) Add reference to ACPDRAFT B.2 explaining why ACP domain membership does not validate ACP address of the connection.

6.4. Downgraded SHOULD to MAY in new -29 suggestion how to deal with DoS attacks with many GRASP announcements. Will also separately ask TSV ADs.

6.4. Fixed extension points in CDDL objective-value definitions (with help from Carsten/Brian).

9.3.5.2. Added explanation when ACP greenfield state ends, and refined text explaining how to deal with this.

11. removed duplicate paragraph (first, kept paragraph was the fixed up, improved correct version).

11. Added references to ACPDRAFT B.1, B.2 as possible future solutions for downgrade attacks.

12. Fixed up text for IANA code point allocation request.

A.6 - removed.

A.9.9 - added one explanatory intro paragraph to makes it easier to distinguish this option from the B.1 considerations.

B.1 - new text suggested from Ben, replacing A.6 (will not be in RFC).

B.2 - new text discussing why there is no network layer address verification in ACP domain membership check (will not be in RFC).

B.4 - Text discussing DULL GRASP attacks via port sweeps and what do do against it.

Other.

1. Added sentence about FCC outage report from June as example for in-band management.

15. added reference to github where document was developed (removed in RFC, part of changelog).

## 15.3. draft-ietf-anima-autonomic-control-plane-29

Comments from Robert Wilton:

Improved several textual nits.

Discuss/Comments from Erik Kline:

Editorial suggestions and nits. Thanks!.

6.1.3 Added text about how/why rsub is irrelevant for domain membership check.

6.3 Added extension points to AN_ACP DULL GRASP objective because for example ACP domain certificate could be a nice optional additional parameter and prior syntax would have forced us to encode into separate objective unnecessarily.

6.7 Using RFC8415 terminology for exponential backoff parameters.

6.11.2 Amended ACP Sub-Addressing table with future code points, explanations and prefix announced into RPL.

6.12.1.11. Reworked text to better explain how black hole route works and added expanation for prefix for manual address scheme.

8.1.3. Reworked explanation of RIOs for ACP connect interfaces for Type C vs. Type A/B hosts.

8.1.4. Added explanation how this "VRF select" option is required for auto-attachment of Type A/B hosts to ACP and other networks.

Discuss/Comments from Barry Leiba:

Various editorial nits - thanks.

6.1 New section pulling in TLS requirements, no need anymore to duplicat for ACP GRASP, EST, BRSKI (ACP/ANI nodes) and (if desired) OCSP/CRLDP. Added rule to start use secure channel only after negotiation has finished. Added rules not to optimize negotiation across multiple L2 interfaces to the same peer.

6.6 Changed role names in secure channel negotiation process: Alice/ Bob -> Decider/Follower. Explanation enhancements. Added definition for ACP nodes with "0" address.

6.8.3 Improved explanation how IKEv2 forces preference of IPsec over GRE due to ACP IPsec profiles being Tunneled vs. Transport.

6.8.4 Limited mentioning of DTLS version requirements to this section.

6.9.2 Removed TLS requirements, they are now in 6.1.

6.10.6 Removed explanation of IANA allocation requirement. Redundant - already in IANA section, and was seen as confusing.

8.1.1 Clarified that there can be security impacts when weakening directly connected address RPF filtering for ACP connect interfaces.

Discuss/Comments from Ben Kaduk:

Many good editorial improvements - thanks!.

5. added explanation of what to do upon failed secure channel establishment.

6.1.1. refined/extended cert public cey crypto algo and better distinguished algo for the keys of the cert and the key of the signer.

6.1.1. and following: explicitly defining "serialNumber" to be the X.520 subject name serialNumber, not the certificate serial Number.

6.1.1. emhasize additional authorization step for EST servers (id-kp-cmcRA).

6.1.2 changed AcpNodeName ABNF to again use 32HEXDIG instead of self-defined variation, because authors overlooked that ABNF is case agnostic (which is fine). Added recommendation to encode as lower case. Added full ABNF encoding for extensions (any characters as in "atoms" except the new "+" separator).

6.1.5.3. New text to explain reason for use of HTTPS (instead of HTTP) for CRLDP and when and how to use HTTPS then.

6.1.5.5. added text explaning why/how and when to maintain TA data upon failing cert renewal (one version with BRSKI, one version with other, ess secure bootstrap protocols).

6.3. new text and requirement about the signaling of transport ports in DULL GRASP - benefits (no well-known ports required), and problems (additional DoS attack vector, albeit not worse than pre-existing ones, depending on setup of L2 subnets.).

6.7.3.1.1. Specified AUTH_HMAC_SHA2_256_128 (as the ESP authentication algorithm).

6.8.2. Added recommendations for TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256 when supporting TLS 1.3.

8.2.2. Added explanation about downgrade attack across configured ACP tunnels and what to do against it.

9.3.5.2. Rewrote most of section as it originally was too centric on BRSKI. Should now well describe expectations against automated bootstrap. Introduces new requirement not to call node as in support of ANI if is ALSO has TOFU bootstrap.

11. Expanded text about malicious EST servers. Added paragraph about ACP secure channel downgrade attacks. Added paragraphs about private PKI as a core to allow security against fake certificates, added paragraph about considerationsproblems when using public PI.

A.10.9 New appendix suggesting how to discover ACP secure channel negotiation downgrade attacks.

Discuss from Roman Danyliw:

6.1.5.1 - Added requirement to only announce SRV.est when a working CA connection.

15 - Amended security considerations with text about registrar dependencies, security of IDevID/ACP-certificate, EST-Server and GRASP for EST server discovery.

Other:

Conversion to XML v3. Solved empty () taxonomy xref problems. Various formatting fixes for v3.

Added contributors section.

## 15.4. draft-ietf-anima-autonomic-control-plane-28

IESG review Roman Danyliw:

6. Requested additional text elaborating misconfiguration plus attack vectors.

6.1.3.1 Added paragraph about unsecured NTP as basis for time in the absence of other options.

6.7.2 reworded text about additional secure channel protocol reqiurements.

6.7.3.1.2. Added requirement for ACP nodes supporting IKEv2 to support RFC8247 (not sure how that got dropped from prior versions.

Replaced minimum crypto requirements definition via specific AES options with more generic "symmetric key/hash strength" requirements.

6.10.7.3. Added example how to derive addressing scheme from IDevID (PID). Added explanation how to deal with non-persistant registrar address database (hint: it sucks or is wasteful, what did you expect).

8.1.1. Added explanation for 'Physical controlled/secured'.

8.1.5. Removed 'Physical controlled/secured' text, refer back to 8.1.1.

8.2.1. Fixed ABNF 'or' syntax line.

9.3.2. Added explanation of remote management problem with interface "down" type commands.

10.2.1. Added explanations for attacks from impaired ACP nodes.

11. Rewrote intro paragraph. Removed text referring to enrollment/ registrars as they are out of scope of ACP (dependencies only).

11. Added note about need for new protocols inside ACP to use end-to-end authentication.

11. Rewrote paragraph about operator mistakes so as to be actionably. Operators must not make mistakes - but ACP minimizes the mistakes they can make.

ACP domain certificate -> ACP certificate.

Various other cosmetic edits (thanks!) and typo fixes (sorry for not running full spell check for every version. Will definitely do before RFC editor).

Other:

6.12.5.2.1./6.12.5.2.2. Added text explaining link breakage wrt. RTL (came about re-analyzing behavior after question about hop count).

Removed now unnecessary references for earlier rrc822Name otherName choice.

### 15.5. draft-ietf-anima-autonomic-control-plane-27

Too many revisions with too many fixes. Lets do a one-word change revision for a change now if it helps to accelerate the review process.

Added "subjectAltName /" to make it unambiguous that AcpNodeName is indeed a SAN (from Russ).

### 15.6. draft-ietf-anima-autonomic-control-plane-26

Russ Housley review of -25.

1.1 Explicit reference for TLS 1.2 RFC.

2. Changed term of "ACP Domain Information" to AcpNodeName (ASN.1) / acp-node-name (ABNF), also through rest of document.

2. Improved CA behavior definition. changed IDevID/LDevID to IDevID/ LDevID certificate to be more unambiguous.

2. Changed definition of root CA to just refer to how its used in RFC7030 CA root key update, because thats the only thing relevant to ACP.

6.1.1 Moved ECDH requirement to end of text as it was not related to the subject of the initial paragraps. Likewise reference to CABFORUM.

6.1.1 Reduced cert key requirements to only be MUST for certs with 2048 RSA public key and P-256 curves. Reduced longer keys to SHOULD.

6.1.2 Changed text for conversion from rfc822Name to otherName / AcpNode, removed all the explanations of benefits coming with rfc822Name *sob* *sob* *sob*.

6.1.2.1 New ASN.1 definition for otherName / AcpNodeName.

6.1.3 Fixed up text. re the handling of missing connectivity for CRLDP / OCSP.

6.1.4 Fixed up text re. inability to use public CA to situation with otherName / AcpNodeName (no more ACME rfc822Name validation for us *sob* *sob* *sob*).

12. Added ASN.1 registration requests to IANA section.

Appenices. Minor changes for rfc822Name to otherName change.

Various minor verbal fixes/enhancements.

## 15.7. draft-ietf-anima-autonomic-control-plane-25

Crypto parameter discuss from Valery Smyslov and Paul Wouters and resulting changes.

6.7.2 Moved Michael Richardson suggested diagnostic of signaling TA from IPsec section to this general requirements section and added explanation how this may be inappropriate if TA payload is considered secret by TA owner.

6.7.3.1 Added traffic selectors for native IPsec. Improved text explanation.

6.7.3.1.2 removed misleading text about signaling TA when using intermediate certs.

6.7.3.1.2 Removed requirement for 'PKCS #7 wrapped X.509 certificate' requirement on request of Valery Smyslov as it is not defined in RFC7296 and there are enough options mandated in RFC7296. Replaced with just informative text to educate readers who are not IPsec experts what the mandatory option in RFC7296 is that allows to signal certificates.

6.7.3.1.2 Added SHOULD requirement how to deal with CERTREQ so that 6.7.2 requirement for TA diagnostics will work in IKEv2 (ignoring CERTREQ is permitted by IKEv2). Added explanation how this will result in TA cert diagnostics.

6.7.3.1.2 Added requirement for IKEv2 to operate on link-local addresses for ACP so at to assume ACT cert as the only possible authenticator - to avoid potentialy failing section from multiple available certs on a router.

6.7.3.1.2 fixed PKIX- style OID to ASN.1 object AlgorithmIdentifier (Paul).

6.7.3.2 Added IPsec traffic selectors for IPsec with GRE.

6.7.5 Added notion that IPsec/GRE MAY be preferred over IPsec/native. Luckily IPsec/native uses tunneling, whereas IPsec/GRE uses transport mode, and there is a long discuss whether it is permitted to even build IPsec connectings that only support transports instead of always being able to fall back to tunnel mode. Added explanatory paragraph why ACP nodes may prefer GRE over native (wonder how that was missing..).

9.1.1 Added section to explain need for secure channel peer diagnostics via signaling of TA. Four examples given.

Paul Wouters mentioned that ipkcs7 had to be used in some interop cases with windows CA, but that is an issue of ACP Registrar having to convert into PKCS#7 to talk to a windows CA, and this spec is not concerned with that, except to know that it is feasible, so not mentioned in text anywhere, just tracking discussion here in changelog.

Michael Richardson:

3.1.3 Added point in support of rfc822address that CA may not support to sign certificates with new attributes (such as new otherName).

Michael Richardson/Brian Carpenter fix:

6.1.5.1/6.3 Fixed GRASP examples.

Joe Halpern review:

1. Enhanded introduction text for in-band and of out-of-band, explaining how ACP is an in-band network aiming to achieve all possible benefits of an out-of-band network.

1. Comprehensive explanation for term Data-Plane as it is only logically following pre-established terminology on a fully autonomic node, when used for existing nodes augmented with ACP, Data-Plane has more functionality than usually associated with the term.

2. Removed explanatory text for Data-Plane, referring to section 1.

2. Reduced explanation in definition of in-band (management/ signaling), out-of-band-signaling, now pointing to section 1.

5. Rewrote a lot of the steps (overview) as this text was not reviewed for long time. Added references to normative section for each step to hopefully avoid feedback of not explaining terms used (really not possible to give good summary without using forward references).

2. Separate out-of-band-management definition from virtual out-of-band-management definition (later one for ACP).

2. Added definitions for RPI and RPL.

6.1.1. added note about end-to-end authentication to distinguish channel security from overall ACP security model.

6.5 Fixed bugs in channel selection signaling step description (Alice vs. Bob).

6.7.1 Removed redundant channel selection explanation.

6.10.3 remove locator/identifier terminology from zone addressing scheme description (unnecessary), removed explanations (now in 9.4), simplified text, clarified requirement for Node-ID to be unique, recommend to use primarily zone 0.

6.10.3.1 Removed. Included a lot of insufficient suggestions for future stanards extensions, most of it was wrong or would need to be revisited by WG anyhow. Idea now (just here for comment): Announce via GRASP Zone-ID (e.g. from per-zone edge-node/registrar) into a zone of the ACP so all nodes supporting the scheme can automatically self-allocate the Zone-ID.

6.11.1.1 (RPL overview), eliminated redundant text.

6.11.1.1.1 New subsection to better structure overview.

6.11.1.1.2 New subsection to better group overview, replaced TTL explanation (just the symptom) with hopefully better reconvergence text (intent of the profile) for the ACP RPL profile.

6.11.1.1.6 Added text to explain simple choice for rank_factor.

6.11.1.13 moved explanation for RPI up into 6.11.1.1.

6.12.5.1 rewrote section for ACP Loopback Interface.

9.4 New informative/informational section for partial or incremental adoption of ACP to help understand why there is the Zone interface sub-scheme, and how to use it.

Unrelated fixes:

Ask to RFC editor to add most important abbreviations to RFC editor abbreviation list.

6.10.2 changed names in ACP addressing scheme table to be less suggestive of use.

Russ Hously review:

2. Fixed definition of "Enrollment", "Trust Anchor", "CA", and "root CA". Changed "Certificate Authority" to "Certification Authority" throughout the document (correct term according to X.509).

6.1 Fixed explanation of mutual ACP trust.

6.1.1 s/X509/X509v3/.

6.1.2 created bulleted lists for explanations and justifications for choices of ACP certificate encoding. No semantic changes, just to make it easier to refer to the points in discussions (rfcdiff seems to have a bug showing text differences due to formatting changes).

6.1.3 Moved content of rule #1 into previous rule #2 because certification chain validation does imply validation of lifetime. numbers of all rules reduced by 1, changed hopefully all references to the rule numbers in the document.

Rule #3, Hopefully fixed linguistic problem self-contradiction of MUST by lower casing MUST in the explanation part and rewriting the condition when this is not applicable.

6.1.4 Replaced redundant term "Trust Point" (TP) with Trust Anchor (TA"). Replaced throughout document Trust Anchor with abbreviation TA.

Enhanced several sentences/rewrote paragraphs to make explanations clearer.

6.6 Added explanation how ACP nodes must throttle their attempts for connection making purely on the result of their own connection attempts, not based on those connections where they are responder.

## 15.8. draft-ietf-anima-autonomic-control-plane-24

Leftover from -23 review by Eric Vyncke:

Swapping sections 9 and 10, section 9 was meant to be at end of document and summarize. Its not meant to be misinterpreted as introducing any new information. This did happen because section 10 was added after section 9.

### 15.9. draft-ietf-anima-autonomic-control-plane-23

Note: big rfcdiff of TOC is an rfcdiff bug, changes really minimal.

Review of IPsec security with Mcr and ipsec mailing list.

6.7.1 - new section: Moved general considerations for secure channel protocols here, refined them.

6.7.2 - new section: Moved common requirements for secure channel protocols here, refined them.

6.7.3.1.1. - improved requirements text related to RFC8221, better explamations re. HW acceleration issues.

6.7.3.1.2. - improved requirements text related to RFC8247, (some requirements still discussed to be redundant, will be finalized in next weeks.

Eric Vyncke review of -21:

Only noting most important changes, long list of smaller text/ readability enhancements.

2. - New explanation of "normative", "informational" section title tags. alphabetic reordering of terms, refined definitions for CA, CRL. root CA.

6.1.1. - explanation when IDevID parameters may be copied into LDevID.

6.1.2. - Fixed hex digits in ACP domain information to lower case.

6.1.3.1. - New section on Realtime clock and Time Validation.

6.3 - Added explanation that DTLS means >= version 1.2 (not only 1.2).

6.7 - New text in this main section explaing relationship of ACP secure channels and ACP virtual interfaces - with forward references to virtual interface section.

6.8.2 - reordered text and picture, no text change.

6.10.7.2 - describe first how Registrar-ID can be allocted for all type of registrars, then refined text for how to potentially use MAC addresses on physical registrars.

6.11.1.1 - Added text how this profile does not use Data-Plane artefacts (RPI) because hadware forwarding. This was previously hidden only later in the text.

6.11.1.13. - Rewrote RPL Data-Plane artefact text. Provide decoder ring for abbreviations and all relevant RFCs.

6.12.5.2. - Added more explicit text that secure channels are mapped into virtual interfaces, moved different type of interfaces used by ACP into separate subsections to be able to refer to them.

7.2 - Rewrote/refined text for ACP on L2, prior text was confusing and did not well explain why ACP for L2/L3 switches can be implemented without any L2 (HW) changes. Also missing explanation of only running GRASP untagged when VLANs are used.

8.1.1 - Added requirement for ACP Edge nodes to allow configurable filtering of IPv6 RPI headers.

11. - (security section). Moved explanation of address stealing from 7.2 to here.

## 15.10. draft-ietf-anima-autonomic-control-plane-22

Ben Kaduk review of -21:

RFC822 encoding of ACP domain information:

6.1.2 rewrote text for explaining / justifying use of rfc822name as identifier for node CP in certificate (was discussed in thread, but badly written in prior versions).

6.1.2 Changed EBNF syntax to use "+" after rfcSELF because that is the known primary name to extensions separator in many email systems ("." was wrong in prior versions).

6.1.2 Rewrote/improved explanations for use of rfc822name field to explain better why it is PKIX compliant and the right thing to do.

Crypto parameters for IPsec:

6.1 - Added explanation of why manual keying for ACP is not feasible for ACP. Surprisingly, that text did not exist. Referred to by IPsec text (6.7.1), but here is the right place to describe the reasoning.

6.1.2 - Small textual refinement referring to requirements to authenticate peers (for the special cases of empty or '0' ACP address in ACP domain information field.

6.3 - To better justify Bens proposed change of secure channel protocol being IPsec vs. GRASP objective being IKEv2, better explained how protocol indicated in GRASP objective-value is name of protocol used to negotiate secure channel, use example of IKEv2 to negotiate IPsec.

6.7.1 - refinemenet similar to 6.3.

- moved new paragraph from Bens pull request up from 6.7.1.1 to 6.7.1 as it equally applies to GRE encapped IPsec (looks nicer one level up).

- created subsections 6.7.1.1 (IPsec/ESP) / 6.7.1.2 (IKEv2) to clearer distinguish between these two requirements blocks.

- Refined the text in these two sections to hopefully be a good answer to Valery's concern of not randomnly mocking with existing requirements docs (rfc8247 / rfc8221).

6.7.1.1.1 - IPsec/ESP requirements section:

- MUST support rfc8221 mandatory EXCEPT for the superceeding requirements in this section. Previously, this was not quite clear from the text.

- Hopefully persuasive explanations about the requirements levels for ENCR_AES_GCM_16, ENCR_AES_CBC, ENCR_AES_CCM_8 and ENCR_CHACHA20_POLY1305: Restructured text for why not ENCR_AES_CBC (was in prior version, just not well structured), added new expanations for ENCR_AES_CCM_8 and ENCR_CHACHA20_POLY130.

- In simple terms, requirements for ENCR_AES_CBC, ENCR_AES_CCM_8, ENCR_CHACHACHA are SHOULD when they are implementable with rqual or faster performancce than ENCR_AES_GCM_16.

- Removed text about "additional rfc8221" reqiurements MAY be used. Now the logic is that all other requirements apply. Hopefully we have written enough so that we prohibited downgrades.

6.7.1.1.2 - RFC8247 requirements:

- Added mandate to support rfc8247, added explanation that there is no "stripping down" requirement, just additional stronger requirements to mandate correct use of ACP certificartes during authentication.

- refined text on identifying ACP by IPv6 address to be clearer: Identifying in the context of IKEv2 and cases for '0' in ACP domain information.

- removed last two paragraphs about relationship to rfc8247, as his is now written in first paragraph of the section.

End of Ben Kaduk review related fixes.

Other:

Forgot to update example of ACP domain information to use capitalized hex-digits as required by HEXDIG used.

Added reference to RFC8316 (AN use-cases) to beginning of section 3 (ACP use cases).

Small Enhanced IPsec parameters description / requirements fixes (from Michael Richardson).

## 16. Normative References

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-bootstrapping-keyinfra-43, 7 August 2020, <http://www.ietf.org/internet-drafts/draft-ietf-anima-bootstrapping-keyinfra-43.txt>.

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", Work in Progress, Internet-Draft, draft-ietf-anima-grasp-15, 13 July 2017, <http://www.ietf.org/internet-drafts/draft-ietf-anima-grasp-15.txt>.

[IKEV2IANA]
IANA, "Internet Key Exchange Version 2 (IKEv2) Parameters", <https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml>.

[RFC1034] Mockapetris, P.V., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[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>.

[RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <https://www.rfc-editor.org/info/rfc3810>.

[RFC4191] Draves, R. and D. Thaler, "Default Router Preferences and More-Specific Routes", RFC 4191, DOI 10.17487/RFC4191, November 2005, <https://www.rfc-editor.org/info/rfc4191>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/info/rfc4193>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[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>.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <https://www.rfc-editor.org/info/rfc4862>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/info/rfc6550>.

[RFC6551] Vasseur, JP., Ed., Kim, M., Ed., Pister, K., Dejean, N., and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks", RFC 6551, DOI 10.17487/RFC6551, March 2012, <https://www.rfc-editor.org/info/rfc6551>.

[RFC6552] Thubert, P., Ed., "Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)", RFC 6552, DOI 10.17487/RFC6552, March 2012, <https://www.rfc-editor.org/info/rfc6552>.

[RFC6553] Hui, J. and JP. Vasseur, "The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams", RFC 6553, DOI 10.17487/RFC6553, March 2012, <https://www.rfc-editor.org/info/rfc6553>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC7676] Pignataro, C., Bonica, R., and S. Krishnan, "IPv6 Support for Generic Routing Encapsulation (GRE)", RFC 7676, DOI 10.17487/RFC7676, October 2015, <https://www.rfc-editor.org/info/rfc7676>.

[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>.

[RFC8221] Wouters, P., Migault, D., Mattsson, J., Nir, Y., and T. Kivinen, "Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 8221, DOI 10.17487/RFC8221, October 2017, <https://www.rfc-editor.org/info/rfc8221>.

[RFC8247] Nir, Y., Kivinen, T., Wouters, P., and D. Migault, "Algorithm Implementation Requirements and Usage Guidance for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 8247, DOI 10.17487/RFC8247, September 2017, <https://www.rfc-editor.org/info/rfc8247>.

[RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <https://www.rfc-editor.org/info/rfc8422>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[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>.

## 17. Informative References

[ACPDRAFT] Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic Control Plane (ACP)", Work in Progress, Internet-Draft, draft-ietf-anima-autonomic-control-plane-30, <https://tools.ietf.org/html/draft-ietf-anima-autonomic-control-plane-30.pdf>. [RFC-Editor: Please remove this complete reference from the RFC] Refer to the IETF working group draft for the few sections removed from this document for various reasons. They capture the state of discussion about unresolved issues that may need to be revisited in future work.

[AR8021] Group, W. -. H. L. L. P. W., "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", December 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

[CABFORUM] CA/Browser Forum, "Certificate Contents for Baseline SSL", November 2019, <https://cabforum.org/baseline-requirements-certificate-contents/>.

[FCC] FCC, "FCC STAFF REPORT ON NATIONWIDE T-MOBILE NETWORK OUTAGE ON JUNE 15, 2020 (PS Docket No. 20-183)", 2020, <https://docs.fcc.gov/public/attachments/DOC-367699A1.docx>. The FCC's Public Safety and Homeland Security Bureau issues a report on a nationwide T-Mobile outage that occurred on June 15, 2020. Action by: Public Safety and Homeland Security Bureau.

[I-D.eckert-anima-noc-autoconfig]
Eckert, T., "Autoconfiguration of NOC services in ACP networks via GRASP", Work in Progress, Internet-Draft, draft-eckert-anima-noc-autoconfig-00, 2 July 2018, <http://www.ietf.org/internet-drafts/draft-eckert-anima-noc-autoconfig-00.txt>.

[I-D.ietf-acme-star]
Sheffer, Y., Lopez, D., Dios, O., Pastor, A., and T. Fossati, "Support for Short-Term, Automatically-Renewed (STAR) Certificates in Automated Certificate Management Environment (ACME)", Work in Progress, Internet-Draft, draft-ietf-acme-star-11, 24 October 2019, <http://www.ietf.org/internet-drafts/draft-ietf-acme-star-11.txt>.

[I-D.ietf-anima-prefix-management]
Jiang, S., Du, Z., Carpenter, B., and Q. Sun, "Autonomic IPv6 Edge Prefix Management in Large-scale Networks", Work in Progress, Internet-Draft, draft-ietf-anima-prefix-

management-07, 18 December 2017, <http://www.ietf.org/internet-drafts/draft-ietf-anima-prefix-management-07.txt>.

[I-D.ietf-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", Work in Progress, Internet-Draft, draft-ietf-anima-reference-model-10, 22 November 2018, <http://www.ietf.org/internet-drafts/draft-ietf-anima-reference-model-10.txt>.

[I-D.ietf-roll-applicability-template]
Richardson, M., "ROLL Applicability Statement Template", Work in Progress, Internet-Draft, draft-ietf-roll-applicability-template-09, 3 May 2016, <http://www.ietf.org/internet-drafts/draft-ietf-roll-applicability-template-09.txt>.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-38, 29 May 2020, <http://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-38.txt>.

[IEEE-1588-2008]
IEEE, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", December 2008, <http://standards.ieee.org/findstds/standard/1588-2008.html>.

[IEEE-802.1X]
Group, W. -. H. L. L. P. W., "IEEE Standard for Local and Metropolitan Area Networks: Port-Based Network Access Control", February 2010, <http://standards.ieee.org/findstds/standard/802.1X-2010.html>.

[LLDP]     Group, W. -. H. L. L. P. W., "IEEE Standard for Local and Metropolitan Area Networks: Station and Media Access Control Connectivity Discovery", June 2016, <https://standards.ieee.org/findstds/standard/802.1AB-2016.html>.

[MACSEC] Group, W. -. H. L. L. P. W., "IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security", June 2006, <https://standards.ieee.org/findstds/standard/802.1AE-2006.html>.

[RFC1112] Deering, S.E., "Host extensions for IP multicasting", STD 5, RFC 1112, DOI 10.17487/RFC1112, August 1989, <https://www.rfc-editor.org/info/rfc1112>.

[RFC1492] Finseth, C., "An Access Control Protocol, Sometimes Called TACACS", RFC 1492, DOI 10.17487/RFC1492, July 1993, <https://www.rfc-editor.org/info/rfc1492>.

[RFC1654] Rekhter, Y., Ed. and T. Li, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 1654, DOI 10.17487/RFC1654, July 1994, <https://www.rfc-editor.org/info/rfc1654>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/info/rfc1918>.

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <https://www.rfc-editor.org/info/rfc2315>.

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, DOI 10.17487/RFC2409, November 1998, <https://www.rfc-editor.org/info/rfc2409>.

[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>.

[RFC3164] Lonvick, C., "The BSD Syslog Protocol", RFC 3164, DOI 10.17487/RFC3164, August 2001, <https://www.rfc-editor.org/info/rfc3164>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <https://www.rfc-editor.org/info/rfc3315>.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, DOI 10.17487/RFC3411, December 2002, <https://www.rfc-editor.org/info/rfc3411>.

[RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", STD 88, RFC 3596, DOI 10.17487/RFC3596, October 2003, <https://www.rfc-editor.org/info/rfc3596>.

[RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, DOI 10.17487/RFC3920, October 2004, <https://www.rfc-editor.org/info/rfc3920>.

[RFC3954] Claise, B., Ed., "Cisco Systems NetFlow Services Export Version 9", RFC 3954, DOI 10.17487/RFC3954, October 2004, <https://www.rfc-editor.org/info/rfc3954>.

[RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, DOI 10.17487/RFC4007, March 2005, <https://www.rfc-editor.org/info/rfc4007>.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <https://www.rfc-editor.org/info/rfc4210>.

[RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <https://www.rfc-editor.org/info/rfc4364>.

[RFC4429] Moore, N., "Optimistic Duplicate Address Detection (DAD) for IPv6", RFC 4429, DOI 10.17487/RFC4429, April 2006, <https://www.rfc-editor.org/info/rfc4429>.

[RFC4541] Christensen, M., Kimball, K., and F. Solensky, "Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches", RFC 4541, DOI 10.17487/RFC4541, May 2006, <https://www.rfc-editor.org/info/rfc4541>.

[RFC4604] Holbrook, H., Cain, B., and B. Haberman, "Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast", RFC 4604, DOI 10.17487/RFC4604, August 2006, <https://www.rfc-editor.org/info/rfc4604>.

[RFC4607] Holbrook, H. and B. Cain, "Source-Specific Multicast for IP", RFC 4607, DOI 10.17487/RFC4607, August 2006, <https://www.rfc-editor.org/info/rfc4607>.

[RFC4610] Farinacci, D. and Y. Cai, "Anycast-RP Using Protocol Independent Multicast (PIM)", RFC 4610, DOI 10.17487/RFC4610, August 2006, <https://www.rfc-editor.org/info/rfc4610>.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <https://www.rfc-editor.org/info/rfc4941>.

[RFC4985] Santesson, S., "Internet X.509 Public Key Infrastructure Subject Alternative Name for Expression of Service Name", RFC 4985, DOI 10.17487/RFC4985, August 2007, <https://www.rfc-editor.org/info/rfc4985>.

[RFC5790] Liu, H., Cao, W., and H. Asaeda, "Lightweight Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Version 2 (MLDv2) Protocols", RFC 5790, DOI 10.17487/RFC5790, February 2010, <https://www.rfc-editor.org/info/rfc5790>.

[RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <https://www.rfc-editor.org/info/rfc5880>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/info/rfc5912>.

[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>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6402] Schaad, J., "Certificate Management over CMS (CMC) Updates", RFC 6402, DOI 10.17487/RFC6402, November 2011, <https://www.rfc-editor.org/info/rfc6402>.

[RFC6407] Weis, B., Rowles, S., and T. Hardjono, "The Group Domain of Interpretation", RFC 6407, DOI 10.17487/RFC6407, October 2011, <https://www.rfc-editor.org/info/rfc6407>.

[RFC6554] Hui, J., Vasseur, JP., Culler, D., and V. Manral, "An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL)", RFC 6554, DOI 10.17487/RFC6554, March 2012, <https://www.rfc-editor.org/info/rfc6554>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/info/rfc6724>.

[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>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <https://www.rfc-editor.org/info/rfc6830>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

[RFC7404] Behringer, M. and E. Vyncke, "Using Only Link-Local Addressing inside an IPv6 Network", RFC 7404, DOI 10.17487/RFC7404, November 2014, <https://www.rfc-editor.org/info/rfc7404>.

[RFC7426] Haleplidis, E., Ed., Pentikousis, K., Ed., Denazis, S., Hadi Salim, J., Meyer, D., and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, DOI 10.17487/RFC7426, January 2015, <https://www.rfc-editor.org/info/rfc7426>.

[RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <https://www.rfc-editor.org/info/rfc7435>.

[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>.

[RFC7585] Winter, S. and M. McCauley, "Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS Based on the Network Access Identifier (NAI)", RFC 7585, DOI 10.17487/RFC7585, October 2015, <https://www.rfc-editor.org/info/rfc7585>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <https://www.rfc-editor.org/info/rfc7721>.

[RFC7761] Fenner, B., Handley, M., Holbrook, H., Kouvelas, I., Parekh, R., Zhang, Z., and L. Zheng, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", STD 83, RFC 7761, DOI 10.17487/RFC7761, March 2016, <https://www.rfc-editor.org/info/rfc7761>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8028] Baker, F. and B. Carpenter, "First-Hop Router Selection by Hosts in a Multi-Prefix Network", RFC 8028, DOI 10.17487/RFC8028, November 2016, <https://www.rfc-editor.org/info/rfc8028>.

[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>.

[RFC8316] Nobre, J., Granville, L., Clemm, A., and A. Gonzalez Prieto, "Autonomic Networking Use Case for Distributed Detection of Service Level Agreement (SLA) Violations", RFC 8316, DOI 10.17487/RFC8316, February 2018, <https://www.rfc-editor.org/info/rfc8316>.

[RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <https://www.rfc-editor.org/info/rfc8366>.

[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>.

[RFC8398] Melnikov, A., Ed. and W. Chuang, Ed., "Internationalized Email Addresses in X.509 Certificates", RFC 8398, DOI 10.17487/RFC8398, May 2018, <https://www.rfc-editor.org/info/rfc8398>.

[RFC8402] Filsfils, C., Ed., Previdi, S., Ed., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", RFC 8402, DOI 10.17487/RFC8402, July 2018, <https://www.rfc-editor.org/info/rfc8402>.

[RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <https://www.rfc-editor.org/info/rfc8572>.

[X.509] International Telecommunication Union, "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks", ITU-T Recommendation X.509, ISO/IEC 9594-8, October 2016, <https://www.itu.int/rec/T-REC-X.509>.

[X.520] International Telecommunication Union, "Information technology - Open Systems Interconnection - The Directory: Selected attribute types", ITU-T Recommendation X.520, ISO/IEC 9594-6, October 2016, <https://www.itu.int/rec/T-REC-X.520>.

# Appendix A. Background and Futures (Informative)

The following sections discuss additional background information about aspects of the normative parts of this document or associated mechanisms such as BRSKI (such as why specific choices were made by the ACP) and they provide discussion about possible future variations of the ACP.

## A.1. ACP Address Space Schemes

This document defines the Zone, Vlong and Manual sub address schemes primarily to support address prefix assignment via distributed, potentially uncoordinated ACP registrars as defined in Section 6.11.7. This costs 48/46-bit identifier so that these ACP registrar can assign non-conflicting address prefixes. This design does not leave enough bits to simultaneously support a large number of nodes (Node-ID) plus a large prefix of local addresses for every node plus a large enough set of bits to identify a routing Zone. In result, Zone, Vlong 8/16 attempt to support all features, but via separate prefixes.

In networks that always expect to rely on a centralized PMS as described above (Section 9.2.5), the 48/46-bits for the Registrar-ID could be saved. Such variations of the ACP addressing mechanisms could be introduced through future work in different ways. If a new otherName was introduced, incompatible ACP variations could be created where every design aspect of the ACP could be changed. Including all addressing choices. If instead a new addressing sub-type would be defined, it could be a backward compatible extension of this ACP specification. Information such as the size of a zone-prefix and the length of the prefix assigned to the ACP node itself could be encoded via the extension field of the acp-node-name.

Note that an explicitly defined "Manual" addressing sub-scheme is always beneficial to provide an easy way for ACP nodes to prohibit incorrect manual configuration of any non-"Manual" ACP address spaces and therefore ensure that "Manual" operations will never impact correct routing for any non-"Manual" ACP addresses assigned via ACP certificates.

## A.2. BRSKI Bootstrap (ANI)

BRSKI describes how nodes with an IDevID certificate can securely and zero-touch enroll with an LDevID certificate to support the ACP. BRSKI also leverages the ACP to enable zero-touch bootstrap of new nodes across networks without any configuration requirements across the transit nodes (e.g., no DHCP/DNS forwarding/server setup). This includes otherwise not configured networks as described in Section 3.2. Therefore, BRSKI in conjunction with ACP provides for a secure and zero-touch management solution for complete networks. Nodes supporting such an infrastructure (BRSKI and ACP) are called ANI nodes (Autonomic Networking Infrastructure), see [I-D.ietf-anima-reference-model]. Nodes that do not support an IDevID certificate but only an (insecure) vendor specific Unique Device Identifier (UDI) or nodes whose manufacturer does not support a MASA could use some future security reduced version of BRSKI.

When BRSKI is used to provision a domain certificate (which is called enrollment), the BRSKI registrar (acting as an enhanced EST server) must include the otherName / AcpNodeName encoded ACP address and domain name to the enrolling node (called pledge) via its response to the pledges EST CSR Attribute request that is mandatory in BRSKI.

The Certification Authority in an ACP network must not change the otherName / AcpNodeName in the certificate. The ACP nodes can therefore find their ACP address and domain using this field in the domain certificate, both for themselves, as well as for other nodes.

The use of BRSKI in conjunction with the ACP can also help to further simplify maintenance and renewal of domain certificates. Instead of relying on CRL, the lifetime of certificates can be made extremely small, for example in the order of hours. When a node fails to connect to the ACP within its certificate lifetime, it cannot connect to the ACP to renew its certificate across it (using just EST), but it can still renew its certificate as an "enrolled/expired pledge" via the BRSKI bootstrap proxy. This requires only that the BRSKI registrar honors expired domain certificates and that the pledge attempts to perform TLS authentication for BRSKI bootstrap using its expired domain certificate before falling back to attempting to use its IDevID certificate for BRSKI. This mechanism could also render CRLs unnecessary because the BRSKI registrar in conjunction with the CA would not renew revoked certificates - only a "Do-not-renew" list would be necessary on BRSKI registrars/CA.

In the absence of BRSKI or less secure variants thereof, provisioning of certificates may involve one or more touches or non-standardized automation. Node vendors usually support provisioning of certificates into nodes via PKCS#7 (see [RFC2315]) and may support this provisioning through vendor specific models via NETCONF ([RFC6241]). If such nodes also support NETCONF Zero-Touch ([RFC8572]) then this can be combined to zero-touch provisioning of domain certificates into nodes. Unless there are equivalent integration of NETCONF connections across the ACP as there is in BRSKI, this combination would not support zero-touch bootstrap across a not configured network though.

### A.3. ACP Neighbor discovery protocol selection

This section discusses why GRASP DULL was chosen as the discovery protocol for L2 adjacent candidate ACP neighbors. The contenders considered where GRASP, mDNS or LLDP.

#### A.3.1. LLDP

LLDP and Cisco's earlier Cisco Discovery Protocol (CDP) are example of L2 discovery protocols that terminate their messages on L2 ports. If those protocols would be chosen for ACP neighbor discovery, ACP neighbor discovery would therefore also terminate on L2 ports. This would prevent ACP construction over non-ACP capable but LLDP or CDP enabled L2 switches. LLDP has extensions using different MAC addresses and this could have been an option for ACP discovery as well, but the additional required IEEE standardization and definition of a profile for such a modified instance of LLDP seemed to be more work than the benefit of "reusing the existing protocol" LLDP for this very simple purpose.

### A.3.2. mDNS and L2 support

Multicast DNNS (mDNS) [RFC6762] with DNS Service Discovery (DNS-SD) Resource Records (RRs) as defined in [RFC6763] is a key contender as an ACP discovery protocol. because it relies on link-local IP multicast, it does operates at the subnet level, and is also found in L2 switches. The authors of this document are not aware of mDNS implementation that terminate their mDNS messages on L2 ports instead of the subnet level. If mDNS was used as the ACP discovery mechanism on an ACP capable (L3)/L2 switch as outlined in Section 7, then this would be necessary to implement. It is likely that termination of mDNS messages could only be applied to all mDNS messages from such a port, which would then make it necessary to software forward any non-ACP related mDNS messages to maintain prior non-ACP mDNS functionality. Adding support for ACP into such L2 switches with mDNS could therefore create regression problems for prior mDNS functionality on those nodes. With low performance of software forwarding in many L2 switches, this could also make the ACP risky to support on such L2 switches.

### A.3.3. Why DULL GRASP

LLDP was not considered because of the above mentioned issues. mDNS was not selected because of the above L2 mDNS considerations and because of the following additional points:

If mDNS was not already existing in a node, it would be more work to implement than DULL GRASP, and if an existing implementation of mDNS was used, it would likely be more code space than a separate implementation of DULL GRASP or a shared implementation of DULL GRASP and GRASP in the ACP.

## A.4. Choice of routing protocol (RPL)

This section motivates why RPL - "IPv6 Routing Protocol for Low-Power and Lossy Networks ([RFC6550] was chosen as the default (and in this specification only) routing protocol for the ACP. The choice and above explained profile was derived from a pre-standard implementation of ACP that was successfully deployed in operational networks.

Requirements for routing in the ACP are:

* Self-management: The ACP must build automatically, without human intervention. Therefore, routing protocol must also work completely automatically. RPL is a simple, self-managing protocol, which does not require zones or areas; it is also self-configuring, since configuration is carried as part of the protocol (see Section 6.7.6 of [RFC6550]).
* Scale: The ACP builds over an entire domain, which could be a large enterprise or service provider network. The routing protocol must therefore support domains of 100,000 nodes or more, ideally without the need for zoning or separation into areas. RPL has this scale property. This is based on extensive use of default routing.
* Low resource consumption: The ACP supports traditional network infrastructure, thus runs in addition to traditional protocols. The ACP, and specifically the routing protocol must have low resource consumption both in terms of memory and CPU requirements. Specifically, at edge nodes, where memory and CPU are scarce, consumption should be minimal. RPL builds a DODAG, where the main resource consumption is at the root of the DODAG. The closer to the edge of the network, the less state needs to be maintained. This adapts nicely to the typical network design. Also, all changes below a common parent node are kept below that parent node.
* Support for unstructured address space: In the Autonomic Networking Infrastructure, node addresses are identifiers, and may not be assigned in a topological way. Also, nodes may move topologically, without changing their address. Therefore, the routing protocol must support completely unstructured address space. RPL is specifically made for mobile ad-hoc networks, with no assumptions on topologically aligned addressing.
* Modularity: To keep the initial implementation small, yet allow later for more complex methods, it is highly desirable that the routing protocol has a simple base functionality, but can import new functional modules if needed. RPL has this property with the concept of "objective function", which is a plugin to modify routing behavior.
* Extensibility: Since the Autonomic Networking Infrastructure is a new concept, it is likely that changes in the way of operation will happen over time. RPL allows for new objective functions to be introduced later, which allow changes to the way the routing protocol creates the DAGs.
* Multi-topology support: It may become necessary in the future to support more than one DODAG for different purposes, using different objective functions. RPL allow for the creation of several parallel DODAGs, should this be required. This could be used to create different topologies to reach different roots.

* No need for path optimization: RPL does not necessarily compute the optimal path between any two nodes. However, the ACP does not require this today, since it carries mainly non-delay-sensitive feedback loops. It is possible that different optimization schemes become necessary in the future, but RPL can be expanded (see point "Extensibility" above).

## A.5. ACP Information Distribution and multicast

IP multicast is not used by the ACP because the ANI (Autonomic Networking Infrastructure) itself does not require IP multicast but only service announcement/discovery. Using IP multicast for that would have made it necessary to develop a zero-touch auto configuring solution for ASM (Any Source Multicast - the original form of IP multicast defined in [RFC1112]), which would be quite complex and difficult to justify. One aspect of complexity where no attempt at a solution has been described in IETF documents is the automatic-selection of routers that should be PIM Sparse Mode (PIM-SM) Rendezvous Points (RPs) (see [RFC7761]). The other aspects of complexity are the implementation of MLD ([RFC4604]), PIM-SM and Anycast-RP (see [RFC4610]). If those implementations already exist in a product, then they would be very likely tied to accelerated forwarding which consumes hardware resources, and that in return is difficult to justify as a cost of performing only service discovery.

Some future ASA may need high performance in-network data replication. That is the case when the use of IP multicast is justified. Such an ASA can then use service discovery from ACP GRASP, and then they do not need ASM but only SSM (Source Specific Multicast, see [RFC4607]) for the IP multicast replication. SSM itself can simply be enabled in the Data-Plane (or even in an update to the ACP) without any other configuration than just enabling it on all nodes and only requires a simpler version of MLD (see [RFC5790]).

LSP (Link State Protocol) based IGP routing protocols typically have a mechanism to flood information, and such a mechanism could be used to flood GRASP objectives by defining them to be information of that IGP. This would be a possible optimization in future variations of the ACP that do use an LSP routing protocol. Note though that such a mechanism would not work easily for GRASP M_DISCOVERY messages which are intelligently (constrained) flooded not across the whole ACP, but only up to a node where a responder is found. We do expect that many future services in ASA will have only few consuming ASA, and for those cases, M_DISCOVERY is the more efficient method than flooding across the whole domain.

Because the ACP uses RPL, one desirable future extension is to use RPLs existing notion of DODAG, which are loop-free distribution trees, to make GRASP flooding more efficient both for M_FLOOD and M_DISCOVERY. See Section 6.13.5 how this will be specifically beneficial when using NBMA interfaces. This is not currently specified in this document because it is not quite clear yet what exactly the implications are to make GRASP flooding depend on RPL DODAG convergence and how difficult it would be to let GRASP flooding access the DODAG information.

## A.6. CAs, domains and routing subdomains

There is a wide range of setting up different ACP solution by appropriately using CAs and the domain and rsub elements in the acp-node-name in the domain certificate. We summarize these options here as they have been explained in different parts of the document in before and discuss possible and desirable extensions:

An ACP domain is the set of all ACP nodes that can authenticate each other as belonging to the same ACP network using the ACP domain membership check (Section 6.2.3). GRASP inside the ACP is run across all transitively connected ACP nodes in a domain.

The rsub element in the acp-node-name permits the use of addresses from different ULA prefixes. One use case is to create multiple physical networks that initially may be separated with one ACP domain but different routing subdomains, so that all nodes can mutual trust their ACP certificates (not depending on rsub) and so that they could connect later together into a contiguous ACP network.

One instance of such a use case is an ACP for regions interconnected via a non-ACP enabled core, for example due to the absence of product support for ACP on the core nodes. ACP connect configurations as defined in this document can be used to extend and interconnect those ACP islands to the NOC and merge them into a single ACP when later that product support gap is closed.

Note that RPL scales very well. It is not necessary to use multiple routing subdomains to scale ACP domains in a way that would be required if other routing protocols where used. They exist only as options for the above mentioned reasons.

If different ACP domains are to be created that should not allow to connect to each other by default, these ACP domains simply need to have different domain elements in the acp-node-name. These domain elements can be arbitrary, including subdomains of one another: Domains "example.com" and "research.example.com" are separate domains if both are domain elements in the acp-node-name of certificates.

It is not necessary to have a separate CA for different ACP domains: an operator can use a single CA to sign certificates for multiple ACP domains that are not allowed to connect to each other because the checks for ACP adjacencies includes comparison of the domain part.

If multiple independent networks choose the same domain name but had their own CA, these would not form a single ACP domain because of CA mismatch. Therefore, there is no problem in choosing domain names that are potentially also used by others. Nevertheless it is highly recommended to use domain names that one can have high probability to be unique. It is recommended to use domain names that start with a DNS domain names owned by the assigning organization and unique within it. For example, "acp.example.com" if you own "example.com".

## A.7. Intent for the ACP

Intent is the architecture component of autonomic networks according to [I-D.ietf-anima-reference-model] that allows operators to issue policies to the network. Its applicability for use is quite flexible and freeform, with potential applications including policies flooded across ACP GRASP and interpreted on every ACP node.

One concern for future definitions of Intent solutions is the problem of circular dependencies when expressing Intent policies about the ACP itself.

For example, Intent could indicate the desire to build an ACP across all domains that have a common parent domain (without relying on the rsub/routing-subdomain solution defined in this document). For example, ACP nodes with domain "example.com", "access.example.com", "core.example.com" and "city.core.example.com" should all establish one single ACP.

If each domain has its own source of Intent, then the Intent would simply have to allow adding the peer domains TA and domain names to the parameters for the ACP domain membership check (Section 6.2.3) so that nodes from those other domains are accepted as ACP peers.

If this Intent was to be originated only from one domain, it could likely not be made to work because the other domains will not build any ACP connection amongst each other, whether they use the same or different CA due to the ACP domain membership check.

If the domains use the same CA one could change the ACP setup to permit for the ACP to be established between two ACP nodes with different acp-domain-names, but only for the purpose of disseminating limited information, such as Intent, but not to set up full ACP connectivity, specifically not RPL routing and passing of arbitrary GRASP information. Unless the Intent policies permit this to happen across domain boundaries.

This type of approach where the ACP first allows Intent to operate and only then sets up the rest of ACP connectivity based on Intent policy could also be used to enable Intent policies that would limit functionality across the ACP inside a domain, as long as no policy would disturb the distribution of Intent. For example, to limit reachability across the ACP to certain type of nodes or locations of nodes.

## A.8. Adopting ACP concepts for other environments

The ACP as specified in this document is very explicit about the choice of options to allow interoperable implementations. The choices made may not be the best for all environments, but the concepts used by the ACP can be used to build derived solutions:

The ACP specifies the use of ULA and deriving its prefix from the domain name so that no address allocation is required to deploy the ACP. The ACP will equally work not using ULA but any other /48 IPv6 prefix. This prefix could simply be a configuration of the ACP registrars (for example when using BRSKI) to enroll the domain certificates - instead of the ACP registrar deriving the /48 ULA prefix from the AN domain name.

Some solutions may already have an auto-addressing scheme, for example derived from existing unique device identifiers (e.g., MAC addresses). In those cases it may not be desirable to assign addresses to devices via the ACP address information field in the way described in this document. The certificate may simply serve to identify the ACP domain, and the address field could be omitted. The only fix required in the remaining way the ACP operate is to define another element in the domain certificate for the two peers to decide who is the Decider and who is the Follower during secure channel building. Note though that future work may leverage the acp address to authenticate "ownership" of the address by the device. If the address used by a device is derived from some pre-existing permanent local ID (such as MAC address), then it would be useful to store that address in the certificate using the format of the access address information field or in a similar way.

The ACP is defined as a separate VRF because it intends to support well managed networks with a wide variety of configurations. Therefore, reliable, configuration-indestructible connectivity cannot be achieved from the Data-Plane itself. In solutions where all transit connectivity impacting functions are fully automated (including security), indestructible and resilient, it would be possible to eliminate the need for the ACP to be a separate VRF. Consider the most simple example system in which there is no separate Data-Plane, but the ACP is the Data-Plane. Add BRSKI, and it becomes a fully autonomic network - except that it does not support automatic addressing for user equipment. This gap can then be closed for example by adding a solution derived from [I-D.ietf-anima-prefix-management].

TCP/TLS as the protocols to provide reliability and security to GRASP in the ACP may not be the preferred choice in constrained networks. For example, CoAP/DTLS (Constrained Application Protocol) may be preferred where they are already used, allowing to reduce the additional code space footprint for the ACP on those devices. Hop-by-hop reliability for ACP GRASP messages could be made to support protocols like DTLS by adding the same type of negotiation as defined in this document for ACP secure channel protocol negotiation. End-to-end GRASP connections can be made to select their transport protocol in future extensions of the ACP meant to better support constrained devices by indicating the supported transport protocols (e.g. TLS/DTLS) via GRASP parameters of the GRASP objective through which the transport endpoint is discovered.

The routing protocol RPL used for the ACP does explicitly not optimize for shortest paths and fastest convergence. Variations of the ACP may want to use a different routing protocol or introduce more advanced RPL profiles.

Variations such as what routing protocol to use, or whether to instantiate an ACP in a VRF or (as suggested above) as the actual Data-Plane, can be automatically chosen in implementations built to support multiple options by deriving them from future parameters in the certificate. Parameters in certificates should be limited to those that would not need to be changed more often than certificates would need to be updated anyhow; Or by ensuring that these parameters can be provisioned before the variation of an ACP is activated in a node. Using BRSKI, this could be done for example as additional follow-up signaling directly after the certificate enrollment, still leveraging the BRSKI TLS connection and therefore not introducing any additional connectivity requirements.

Last but not least, secure channel protocols including their encapsulations are easily added to ACP solutions. ACP hop-by-hop network layer secure channels could also be replaced by end-to-end security plus other means for infrastructure protection. Any future network OAM should always use end-to-end security anyhow and can leverage the domain certificates and is therefore not dependent on security to be provided for by ACP secure channels.

### A.9. Further (future) options

#### A.9.1. Auto-aggregation of routes

Routing in the ACP according to this specification only leverages the standard RPL mechanism of route optimization, e.g. keeping only routes that are not towards the RPL root. This is known to scale to networks with 20,000 or more nodes. There is no auto-aggregation of routes for /48 ULA prefixes (when using rsub in the acp-node-name) and/or Zone-ID based prefixes.

Automatic assignment of Zone-ID and auto-aggregation of routes could be achieved for example by configuring zone-boundaries, announcing via GRASP into the zones the zone parameters (zone-ID and /48 ULA prefix) and auto-aggregating routes on the zone-boundaries. Nodes would assign their Zone-ID and potentially even /48 prefix based on the GRASP announcements.

#### A.9.2. More options for avoiding IPv6 Data-Plane dependencies

As described in Section 6.13.2, the ACP depends on the Data-Plane to establish IPv6 link-local addressing on interfaces. Using a separate MAC address for the ACP allows to fully isolate the ACP from the Data-Plane in a way that is compatible with this specification. It is also an ideal option when using Single-root input/output virtualization (SR-IOV - see https://en.wikipedia.org/wiki/Single-root_input/output_virtualization) in an implementation to isolate the ACP because different SR-IOV interfaces use different MAC addresses.

When additional MAC address(es) are not available, separation of the ACP could be done at different demux points. The same subnet interface could have a separate IPv6 interface for the ACP and Data-Plane and therefore separate link-local addresses for both, where the ACP interface is non-configurable on the Data-Plane. This too would be compatible with this specification and not impact interoperability.

An option that would require additional specification is to use a different Ethertype from 0x86DD (IPv6) to encapsulate IPv6 packets for the ACP. This would be a similar approach as used for IP authentication packets in [IEEE-802.1X] which use the Extensible Authentication Protocol over Local Area Network (EAPoL) ethertype (0x88A2).

Note that in the case of ANI nodes, all the above considerations equally apply to the encapsulation of BRSKI packets including GRASP used for BRSKI.

### A.9.3. ACP APIs and operational models (YANG)

Future work should define YANG ([RFC7950]) data model and/or node internal APIs to monitor and manage the ACP.

Support for the ACP Adjacency Table (Section 6.3) and ACP GRASP need to be included into such model/API.

### A.9.4. RPL enhancements

```
      ..... USA ......              ..... Europe ......

           NOC1                           NOC2
            |                              |
            |            metric 100        |
          ACP1 --------------------------- ACP2  .
            |                              |     . WAN
            | metric 10         metric 20  |     . Core
            |                              |     .
          ACP3 --------------------------- ACP4  .
            |            metric 100        |
            |                              |     .
            |                              |     . Sites
          ACP10                           ACP11  .
```

Figure 19: Dual NOC

The profile for RPL specified in this document builds only one spanning-tree path set to a root, typically a registrar in one NOC. In the presence of multiple NOCs, routing toward the non-root NOCs may be suboptimal. Figure 19 shows an extreme example. Assuming that node ACP1 becomes the RPL root, traffic between ACP11 and NOC2 will pass through ACP4-ACP3-ACP1-ACP2 instead of ACP4-ACP2 because the RPL calculated DODAG/routes are shortest paths towards the RPL root.

To overcome these limitations, extensions/modifications to the RPL profile can provide optimality for multiple NOCs. This requires utilizing Data-Plane artifact including IPinIP encap/decap on ACP routers and processing of IPv6 RPI headers. Alternatively, (Src,Dst) routing table entries could be used.

Flooding of ACP GRASP messages can be further constrained and therefore optimized by flooding only via links that are part of the RPL DODAG.

### A.9.5. Role assignments

ACP connect is an explicit mechanism to "leak" ACP traffic explicitly (for example in a NOC). It is therefore also a possible security gap when it is easy to enable ACP connect on arbitrary compromised ACP nodes.

One simple solution is to define an extension in the ACP certificates ACP information field indicating the permission for ACP connect to be configured on that ACP node. This could similarly be done to decide whether a node is permitted to be a registrar or not.

Tying the permitted "roles" of an ACP node to the ACP certificate provides fairly strong protection against misconfiguration, but is still subject to code modifications.

Another interesting role to assign to certificates is that of a NOC node. This would allow to limit certain type of connections such as OAM TLS connections to only NOC initiator or responders.

### A.9.6. Autonomic L3 transit

In this specification, the ACP can only establish autonomic connectivity across L2 hops and only explicitly configured options to tunnel across L3. Future work should specify mechanisms to automatically tunnel ACP across L3 networks. A hub&spoke option would allow to tunnel across the Internet to a cloud or central instance of the ACP, a peer-to-peer tunneling mechanism could tunnel ACP islands across an L3VPN infrastructure.

### A.9.7. Diagnostics

Section 9.1 describes diagnostics options that can be done without changing the external, interoperability affecting characteristics of ACP implementations.

Even better diagnostics of ACP operations is possible with additional signaling extensions, such as:

1. Consider if LLDP should be a recommended functionality for ANI devices to improve diagnostics, and if so, which information elements it should signal (noting that such information is conveyed in an insecure manner). Includes potentially new information elements.
2. In alternative to LLDP, A DULL GRASP diagnostics objective could be defined to carry these information elements.
3. The IDevID certificate of BRSKI pledges should be included in the selected insecure diagnostics option. This may be undesirable when exposure of device information is seen as too much of a security issue (ability to deduce possible attack vectors from device model for example).
4. A richer set of diagnostics information should be made available via the secured ACP channels, using either single-hop GRASP or network wide "topology discovery" mechanisms.

### A.9.8. Avoiding and dealing with compromised ACP nodes

Compromised ACP nodes pose the biggest risk to the operations of the network. The most common type of compromise is leakage of credentials to manage/configure the device and the application of malicious configuration including the change of access credentials, but not the change of software. Most of today's networking equipment should have secure boot/software infrastructure anyhow, so attacks that introduce malicious software should be a lot harder.

The most important aspect of security design against these type of attacks is to eliminate password based configuration access methods and instead rely on certificate based credentials handed out only to nodes where it is clear that the private keys cannot leak. This limits unexpected propagation of credentials.

If password based credentials to configure devices still need to be supported, they must not be locally configurable, but only be remotely provisioned or verified (through protocols like RADIUS or Diameter), and there must be no local configuration permitting to change these authentication mechanisms, but ideally they should be autoconfiguring across the ACP. See [I-D.eckert-anima-noc-autoconfig].

Without physical access to the compromised device, attackers with access to configuration should not be able to break the ACP connectivity, even when they can break or otherwise manipulate (spoof) the Data-Plane connectivity through configuration. To achieve this, it is necessary to avoid providing configuration options for the ACP, such as enabling/disabling it on interfaces. For example, there could be an ACP configuration that locks down the current ACP config unless factory reset is done.

With such means, the valid administration has the best chances to maintain access to ACP nodes, discover malicious configuration though ongoing configuration tracking from central locations for example, and to react accordingly.

The primary reaction is withdrawal/change of credentials, terminate malicious existing management sessions and fixing the configuration. Ensuring that management sessions using invalidated credentials are terminated automatically without recourse will likely require new work.

Only when these steps are not feasible would it be necessary to revoke or expire the ACP certificate credentials and consider the node kicked off the network - until the situation can be further rectified, likely requiring direct physical access to the node.

Without extensions, compromised ACP nodes can only be removed from the ACP at the speed of CRL/OCSP information refresh or expiry (and non-removal) of short lived certificates. Future extensions to the ACP could for example use GRASP flooding distribution of triggered updates of CRL/OCSP or explicit removal indication of the compromised nodes domain certificate.

### A.9.9. Detecting ACP secure channel downgrade attacks

The following text proposes a mechanism to protect against downgrade attacks without introducing a new specialized UPFRONT GRASP secure channel mechanism. Instead, it relies on running GRASP after establishing a secure channel protocol to verify if the established secure channel option could have been the result of a MITM downgrade attack:

MITM attackers can force downgrade attacks for ACP secure channel selection by filtering/modifying DULL GRASP messages and/or actual secure channel data packets. For example, if at some point in time DTLS traffic could be easier decrypted than traffic of IKEv2, the MITM could filter all IKEv2 packets to force ACP nodes to use DTLS (assuming the ACP nodes in question supported both DTLS and IKEv2).

For cases where such MITM attacks are not capable to inject malicious traffic (but only to decrypt the traffic), a downgrade attack could be discovered after a secure channel connection is established, for example by use of the following type of mechanism:

After the secure channel connection is established, the two ACP peers negotiate via an appropriate (To Be Defined) GRASP negotiation which ACP secure channel protocol should have been selected between them (in the absence of a MITM attacker). This negotiation would have to signal the DULL GRASP announced ACP secure channel options by each peer followed by an announcement of the preferred secure channel protocol by the ACP peer that is the Decider in the secure channel setup, e.g. the ACP peer that is deciding which secure channel protocol to pick. If that chosen secure channel protocol is different from the one that actually was chosen, then this mismatch is an indication that there is a MITM attacker or other similar issue (firewall prohibiting the use of specific protocols) that caused a non-preferred secure channel protocol to be chosen. This discovery could then result in mitigation options such as logging and ensuing investigations.

## Appendix B. Unfinished considerations (To Be Removed From RFC)

[RFC-Editor: This whole appendix B. and its subsections to be removed for the RFC.

This appendix contains unfinished considerations that are removed from the RFC, they are maintained in this draft as a log of the state of discussion and point of reference. Together with this appendix, also the references pointing to it are marked to be removed from the RFC because no consensus could be reached that a self-reference to a draft version of the RFC is an appropriate breadcrumb to point to unfinished considerations.

The authors plan to move these considerations into a new target informational draft, please look for draft-eckert-anima-acp-considerations.

### B.1. Considerations for improving secure channel negotiation

Proposed text from Benjamin Kaduk. It is suggested to replace the text of appendix A.6 in previous versions of this draft (up to version 29).

The discovery procedure in this specification for low-level ACP channel support by layer-2 peers involves DULL GRASP and attempting (usually in parallel) to establish all supported channel types, learning the peer ACP address and correspondingly the assignment of Decider and Follower roles, and tearing down all channels other than the one preferred by the Decider. This procedure, in general, becomes resource intensive as the number of possible secure channels grows; even worse, under some threat models, the security of the discovery result is only as strong as the weakest supported secure channel protocol. Furthermore, the unilateral determination of "best" channel type by the Decider does not result in the optimal outcome in all possible scenarios.

This situation is tolerable at present, with only two secure channels (DTLS and IPsec) defined, but long-term agility in the vein of [BCP201] will require the introduction of an alternate discovery/negotiation procedure. While IKEv2 is the IETF standard protocol for negotiating security associations, it currently does not have a defined mechanism flexible enough to negotiate the parameters needed for, e.g., an ACP DTLS channel, let alone for allowing ACP peers to indicate their preference metrics for channel selection. Such a mechanism or mechanisms could be defined, but if ACP agility requires introducing a new channel type, for example MacSec, IKEv2 would again need to be extended in order to negotiate an ACP MacSec association. Making ACP channel agility dependent on updates to IKEv2 is likely to result in obstacles due to different timescales of evolution, since IKEv2 implementations help form the core of Internet-scale security infrastructure and must accordingly be robust and thoroughly tested.

Accordingly, a dedicated ACP channel negotiation mechanism is appropriate as a way to provide long-term algorithm and secure-channel protocol agility. Such a mechanism is not currently defined, but one possible design is as follows. A new DULL GRASP objective is defined to indicate the GRASP-over-TLS channel, which is by definition preferred to other channel types (including DTLS and IPsec). When both peers advertise support for GRASP-over-TLS, GRASP-over-TLS must run to completion before other channel types are attempted. The GRASP-over-TLS channel performs the necessary negotiation by establishing a TLS connection between the peers and using that connection to secure a dedicated GRASP instance for negotiating supported channel types and preference metrics. This provides a rich language for determining what secure channel protocol to use for the ACP link while taking into account the capabilities and preferences of the ACP peers, all protected by the security of the TLS channel.

## B.2. ACP address verification

The AcpNodeName of most ACP nodes contains in the acp-address field the primary ACP address to be used by the node for end-to-end connections across ACP secure channels. Nevertheless, there is no verification of an ACP peers address specified in this document. This section explains the current understanding as to why this is not done.

Not all ACP nodes will have an actual IPv6 address in the acp-address field of their AcpNodeName. Those who do not include nodes that do not support ACP secure channels, such as pre-existing NOC equipment that only connects to the ACP via ACP connect interfaces. Likewise, future ACP node type that may want to have their Node-ID not be defined by an ACP registrar, but differently cannot have the ACP address be provided in their ACP certificate where it would be defined by the registrar. In result, any scheme that would rely on verification of the acp-address in the ACP certificate would only apply to a subset of ACP nodes.

The transport stack network layer address used for ACP secure channels is not the acp-address. For automatically established ACP secure channels, it is a link-local IPv6 address. For explicitly configured ACP secure channels (to reach across non ACP L3 network segments), the address is any IPv4 or IPv6 address routable to that remote destination.

When the acp-address is actually used across the ACP, it can only be verified by a peer when the peer has the certificate of the peer. Unless further higher layer mechanisms are developed on top of the ACP (for example via ASA), the only mechanism to access a peers ACP certificate is for secure connections in which the peers certificates are exchanged and cryptographically verified, e.g. TLS and DTLS. Initially, it is expected that the ACP will carry many legacy network management control connections that unfortunately not end-to-end authenticated but that are solely protected by being carried across the ACP secure channels. ACP address verification therefore cannot be used for such connections without additional higher layer components.

For the remaining (TLS/DTLS) connections for which address verification can be used, the main question is: what additional benefit would address verification provide?

The main value that transport stack network layer address verification could provide for these type of connections would be the discovery of on-path transport proxies. For example, in case of BRSKI, pledges connect to an ACP registrar via an ASA implementing a TCP proxy because the pledge itself has at that point in time no ACP certificate valid to build ACP secure channels and hence needs to rely on such a proxy. This is one example where such a TCP proxy is required and not a form of attack.

In general, on path TCP proxies could be a form of attack, but it stands to reason, that an attacker that manages to enable a malicious TCP proxy could likely equally build a transparent proxy not changing the network layer addresses. Only when the attacker operates off-path would this option not be possible. Such attacks could indeed be possible: An impaired ACP node could announce itself as another service instance for a service whose utilization it wants to attack. It could then attempt to look like a valid server by simply TCP proxying the clients connections to a valid server and then attack the connections passing through it (passive decrypting or passive fingerprint analysis). But like the BRSKI proxy, this behavior could be perfectly legitimate and not an attack. For example, TCP has in the past often suffered from performance issues across difficult (high capacity, high loss) paths, and TCP proxies where and are being used simply as a tool for isolating such path segments (such as a WAN), and providing caching and local-retransmit of in-transit packets, reducing the effective path segment capacity.

As explained elsewhere in this document already, considerations for these type of attack are therefore outside the scope of the ACP but fundametal to further design of the ASA infrastructure. Beyond distinguishing whether a TCP proxy would be beneficial or malicious, the even more fundamental question is how to determine from a multitude of service announcements which instance is the most trustworthy and functionally best. In the Internet/web, this question is NOT solved inside the network but through off-net human interaction ("trust me, the best search engine is www.<insert-your-personal-recommendation>.com").

## B.3. Public CA considerations

Public CAs are outside the scope of this document for the following reasons. This appendix describes the current state of understanding for those interested to consider utilizing public CA for the ACP in the future.

If public CA where to be used to enroll ACP nodes and act as TA, this would require a model in which the public CA would be able to assert the ownership of the information requested in the certificate, especially the AcpNodeName, for example mitigated by the domain registrar(s). Due to the use of a new, ACP unique encoding of the AcpNodeName, there is no mechanism for public CA to do so. More importantly though, isolation between ACPs of disjoint operated ACPs is achieved in the current ACP design through disjoint TA. A public CA is in general based on a single (set of) TA shared across all certificates signed by the CA.

Due to the fact that the ACP domain membership check also validates that a peers domain name in the AcpNodeName matches that of the ACP node itself, it would be possible to use the same TA across disjoint ACP domains, but the security and attack implications of such an approach are beyond the scope of this document.

The use of ULA addresses in the AcpNodeName is another novel aspect for certificates from a possible public CA. Typically, ULA addresses are not meant to be signed by a public CA when carried in an address field, because there is no ownership of a particular ULA address in the scope of the Internet, which is what public CA operate on.

Nevertheless, the ULA addresses used by the ACP are scoped to be valid only within the confines of a specific ACP as defined by the domain name in the AcpNodeName. However, this understanding has not been reviewed or accepted by any bodies responsible for policies of public CA.

Because in this specification, ACPs are isolated from each other primarily by their TA, when a public CA would intend to sign ACP certificates and using a single TA to sign TA of ACP certificates from different operators/domain, it could do so by ensuring that the domain name in the AcpNodeName was a globally owned DNS ACP domain name of the organization, and beyond that, it would need to validate that the ACP registrar of that domain who is mitigating the enrollment is authorized to vouch for the ownership of the acp-address within the scope of the ACP domain name.

## B.4. Hardening DULL GRASP considerations

DULL GRASP suffers from similar type of DoS attacks as many link-local multicast discovery protocols, for example mDNS. Attackers on a subnet may be able to inject malicious DULL GRASP messages that are indistinguishable from non-malicious DULL GRASP messages to create Denial-of-Service (DoS) attacks that force ACP nodes to attempt many unsuccessful ACP secure channel connections.

When an ACP node sees multiple AN_ACP objectives for the same secure channel protocol on different transport addresses, it could prefer connecting via the well-known transport address if the secure channel method has one, such as UDP port 500 for IKEv2. For protocols such as (ACP secure channel over) DTLS for which there are no well defined port number, this heuristic does not provide benefits though.

DoS attack with port numbers can also be eliminated by relying on well known-port numbers implied by the GRASP method-name. For example, a future service name of "DTLSacp" could be defined to be associated only to a newly to be assigned well known UDP port for ACP over DTLS, and the port number in the GRASP transport address information would be ignored. Note that there is already a variety of ports assigned to specific protocols over DTLS by IANA, so especially for DTLS this would not be uncommon.

## Authors' Addresses

Toerless Eckert (editor)
Futurewei Technologies Inc. USA
2330 Central Expy
Santa Clara,  95050
United States of America

Email: tte+ietf@cs.fau.de

Michael H. Behringer (editor)

Email: michael.h.behringer@gmail.com

Steinthor Bjarnason
Arbor Networks
2727 South State Street, Suite 200
Ann Arbor,  MI 48104
United States

Email: sbjarnason@arbor.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 18 May 2021

B. E. Carpenter
Univ. of Auckland
B. Liu, Ed.
Huawei Technologies
W. Wang
X. Gong
BUPT University
14 November 2020

# Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)

draft-ietf-anima-grasp-api-08

## Abstract

This document is a conceptual outline of an application programming interface (API) for the Generic Autonomic Signaling Protocol (GRASP). Such an API is needed for Autonomic Service Agents (ASA) calling the GRASP protocol module to exchange autonomic network messages with other ASAs. Since GRASP is designed to support asynchronous operations, the API will need to be adapted to the support for asynchronicity in various programming languages and operating systems.

## Status of This Memo



## Copyright Notice

Table of Contents



## 1. Introduction

As defined in [I-D.ietf-anima-reference-model], the Autonomic Service Agent (ASA) is the atomic entity of an autonomic function, and it is instantiated on autonomic nodes. These nodes are members of a secure Autonomic Control Plane (ACP) such as defined by [I-D.ietf-anima-autonomic-control-plane].

When ASAs communicate with each other, they should use the Generic Autonomic Signaling Protocol (GRASP) [I-D.ietf-anima-grasp]. GRASP relies on the message confidentiality and integrity provided by the

ACP, with the consequence that all nodes in a given autonomic network share the same trust boundary, i.e., the boundary of the ACP. Nodes that have not successfully joined the ACP cannot send, receive or intercept GRASP messages via the ACP, and cannot usurp ACP addresses. An ASA runs in an ACP node and therefore inherits all its security properties, i.e., message integrity, message confidentiality and the fact that unauthorized nodes cannot join the ACP. All ASAs within a given autonomic network therefore trust each other's messages. For these reasons, the API defined in this document has no explicit security features.

An important feature of GRASP is the concept of a GRASP Objective. This is a data structure whose main contents are a name and a value, explained at more length in the 'Terminology' section of [I-D.ietf-anima-grasp]. When an Objective is passed from one ASA to another using GRASP, its value is either conveyed in one direction (by a process of synchronization or flooding), or negotiated bilaterally. The semantics of the value are opaque to GRASP and therefore to the API. Each objective must be accurately specified, as discussed in the 'Objective Options' section of [I-D.ietf-anima-grasp]. Data storage and consistency during negotiation are the responsibility of the ASAs involved. Additionally, GRASP needs to cache the latest values of Objectives that are received by flooding.

As Figure 1 shows, a GRASP implementation could contain several sub-layers. The bottom layer is the GRASP base protocol module, which is only responsible for sending and receiving GRASP messages and maintaining shared data structures. Above that is the basic API described in this document. The upper layer contains some extended API functions based upon GRASP basic protocol. For example, [I-D.ietf-anima-grasp-distribution] describes a possible extended function.

Multiple ASAs in a single node will share the same instance of GRASP, much as multiple applications share a single TCP/IP stack. This aspect is hidden from individual ASAs by the API, and is not further discussed here.

It is desirable that ASAs can be designed as portable user-space programs using a system-independent API. In many implementations, the GRASP code will therefore be split between user space and kernel space. In user space, library functions provide the API and communicate directly with ASAs. In kernel space is a daemon, or a set of sub-services, providing GRASP core functions that are independent of specific ASAs, such as multicast handling and relaying, and common data structures such as the discovery cache. The GRASP API library would need to communicate with the GRASP core via an inter-process communication (IPC) mechanism. The details of this are system-dependent.

```
                +--------------+          +--------------+
                |     ASAs     |          |     ASAs     |
                +--------------+          +--------------+
                  |          |                      |
                  | +-------------------+           |
                  | | GRASP Extended    |           |
                  | | Function API      |           |
                  | +-------------------+           |
                  |          |                      |
               +-------------------------------------------+
               |             GRASP API Library             |
               +-------------------------------------------+
                                 |
                                IPC
                                 |
               +-------------------------------------------+
               |  GRASP Core                               |
               |  (functions, data structures, daemon(s))  |
               +-------------------------------------------+
```

Figure 1: Software layout

Both the GRASP library and the extended function modules should be available to the ASAs. However, since the extended functions are expected to be added in an incremental manner, they will be the subject of future documents. This document only describes the basic GRASP API.

The functions provided by the API do not map one-to-one onto GRASP messages. Rather, they are intended to offer convenient support for message sequences (such as a discovery request followed by responses from several peers, or a negotiation request followed by various possible responses). This choice was made to assist ASA programmers in writing code based on their application requirements rather than needing to understand protocol details.

Note that a simple autonomic node might contain very few ASAs in addition to the autonomic infrastructure components described in [I-D.ietf-anima-bootstrapping-keyinfra] and [I-D.ietf-anima-autonomic-control-plane]. Such a node might directly integrate a GRASP protocol stack in its code and therefore not require this API to be installed. However, the programmer would then need a deeper understanding of the GRASP protocol than is needed to use the API.

This document gives a conceptual outline of the API. It is not a formal specification for any particular programming language or operating system, and it is expected that details will be clarified in individual implementations.

## 2. GRASP API for ASA

### 2.1. Design Assumptions

The assumption of this document is that any Autonomic Service Agent (ASA) needs to call a GRASP module. The latter handles protocol details (security, sending and listening for GRASP messages, waiting, caching discovery results, negotiation looping, sending and receiving sychronization data, etc.) but understands nothing about individual GRASP objectives (Section 2.10 of [I-D.ietf-anima-grasp]). The semantics of objectives are unknown to the GRASP module and are handled only by the ASAs. Thus, this is an abstract API for use by ASAs. Individual language bindings should be defined in separate documents.

Different ASAs may make different use of GRASP features:

* Use GRASP only for discovery purposes.

* Use GRASP negotiation but only as an initiator (client).

* Use GRASP negotiation but only as a responder.

* Use GRASP negotiation as an initiator or responder.

* Use GRASP synchronization but only as an initiator (recipient).

* Use GRASP synchronization but only as a responder and/or flooder.

* Use GRASP synchronization as an initiator, responder and/or flooder.

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

The API design assumes that the operating system and programming language provide a mechanism for simultaneous asynchronous operations. This is discussed in detail in Section 2.2.

A few items are out of scope in this version, since practical experience is required before including them:

* Authorization of ASAs is not defined as part of GRASP and is not supported.

* User-supplied explicit locators for an objective are not supported. The GRASP core will supply the locator, using the ACP address of the node concerned.

* The Rapid mode of GRASP (Section 2.5.4 of [I-D.ietf-anima-grasp]) is not supported.

## 2.2. Asynchronous Operations

GRASP depends on asynchronous operations and wait states, and some of its messages are not idempotent, meaning that repeating a message may cause repeated changes of state in the recipient ASA. Many ASAs will need to support several concurrent operations; for example an ASA might need to negotiate one objective with a peer while discovering and synchronizing a different objective with a different peer. Alternatively, an ASA which acts as a resource manager might need to run simultaneous negotiations for a given objective with multiple different peers. Such an ASA will probably need to support uninterruptible atomic changes to its internal data structures, using a mechanism provided by the operating system and programming language in use.

### 2.2.1. Alternative Asynchronous Mechanisms

Thus, some ASAs need to support asynchronous operations, and therefore the GRASP core must do so. Depending on both the operating system and the programming language in use, there are various techniques for such parallel operations, three of which we consider here: multi-threading, an event loop structure using polling, and an event loop structure using callback functions.

1. In multi-threading, the operating system and language will provide the necessary support for asynchronous operations, including creation of new threads, context switching between

threads, queues, locks, and implicit wait states. In this case, API calls can be treated as simple synchronous function calls within their own thread, even if the function includes wait states, blocking and queueing. Concurrent operations will each run in their own threads. For example, the discover() call may not return until discovery results have arrived or a timeout has occurred. If the ASA has other work to do, the discover() call must be in a thread of its own.

2. In an event loop implementation with polling, blocking calls are not acceptable. Therefore all calls must be non-blocking, and the main loop could support multiple GRASP sessions in parallel by repeatedly polling each one for a change of state. To facilitate this, the API implementation would provide non-blocking versions of all the functions that otherwise involve blocking and queueing. In these calls, a 'noReply' code will be returned by each call instead of blocking, until such time as the event for which it is waiting (or a failure) has occurred. Thus, for example, discover() would return 'noReply' instead of waiting until discovery has succeeded or timed out. The discover() call would be repeated in every cycle of the main loop until it completes. Effectively, it becomes a polling call.

3. It was noted earlier that some GRASP messages are not idempotent; in particular this applies to each step in a negotiation session - sending the same message twice might produce unintended side effects. This is not affected by event loop polling: repeating a call after a 'noReply' does not repeat a message; it simply checks whether a reply has been received.

4. In an event loop implementation with callbacks, the ASA programmer would provide a callback function for each asynchronous operation, e.g. discovery_received(). This would be called asynchronously when a reply is received or a failure such as a timeout occurs.

The following calls involve waiting for a remote operation, so they could use a polling or callback mechanism. In a threaded mechanism, they will usually require to be called in a separate thread:

discover() whose callback would be discovery_received().

request_negotiate() whose callback would be negotiate_step_received().

negotiate_step() whose callback would be negotiate_step_received().

listen_negotiate() whose callback would be negotiate_step_received().

synchronize() whose callback would be synchronization_received().

### 2.2.2. Multiple Negotiation Scenario

The design of GRASP allows the following scenario. Consider an ASA "A" that acts as a resource allocator for some objective. An ASA "B" launches a negotiation with "A" to obtain or release a quantity of the resource. While this negotatition is under way, "B" chooses to launch a second simultaneous negotiation with "A" for a different quantity of the same resource. "A" must therefore conduct two separate negotiation sessions at the same time with the same peer, and must not mix them up.

Note that ASAs could be designed to avoid such a scenario, i.e. restricted to exactly one negotiation session at a time for a given objective, but this would be a voluntary restriction not required by the GRASP protocol. In fact it is an assumption of GRASP that any ASA managing a resource may need to conduct multiple parallel negotiations, possibly with the same peer. Communication patterns could be very complex, with a group of ASAs overlapping negotiations among themselves, as described in [I-D.ciavaglia-anima-coordination]. Therefore, the API design allows for such scenarios.

In the callback model, for the scenario just described, the ASAs "A" and "B" will each provide two instances of negotiate_step_received(), one for each session. For this reason, each ASA must be able to distinguish the two sessions, and the peer's IP address is not sufficient for this. It is also not safe to rely on transport port numbers for this, since future variants of GRASP might use shared ports rather than a separate port per session. Hence the GRASP design includes a session identifier. Thus, when necessary, a 'session_nonce' parameter is used in the API to distinguish simultaneous GRASP sessions from each other, so that any number of sessions may proceed asynchronously in parallel.

### 2.2.3. Overlapping Sessions and Operations

A GRASP session consists of a finite sequence of messages (for discovery, synchronization, or negotiation) between ASAs. It is identified by a pseudo-random session identifier tagged with an IP address of the initiator of the session to guarantee uniqueness. Further details are given in [I-D.ietf-anima-grasp].

On the first call in a new GRASP session, the API returns a 'session_nonce' value based on the GRASP session identifier. This value must be used in all subsequent calls for the same session, and will be provided as a parameter in the callback functions. By this mechanism, multiple overlapping sessions can be distinguished, both in the ASA and in the GRASP core. The value of the 'session_nonce" is opaque to the ASA.

An additional mechanism that might increase efficiency for polling implementations is to add a general call, say notify(), which would check the status of all outstanding operations for the calling ASA and return the session_nonce values for all sessions that have changed state. This would eliminate the need for repeated calls to the individual functions returning a 'noReply'. This call is not described below as the details are likely to be implementation-specific.

An implication of the above for all GRASP implementations is that the GRASP core must keep state for each GRASP operation in progress, most likely keyed by the GRASP Session ID and the GRASP source address of the session initiator. Even in a threaded implementation, the GRASP core will need such state internally. The session_nonce parameter exposes this aspect of the implementation.

### 2.3. API definition

#### 2.3.1. Overview of Functions

The functions provided by the API fall into several groups:

* Registration. These functions allow an ASA to register itself with the GRASP core, and allow a registered ASA to register the GRASP Objectives that it will manipulate.

* Discovery. This function allows an ASA that needs to initiate negotiation or synchronization of a particular Objective to discover a peer willing to respond.

* Negotiation. These functions allow an ASA to act as an initiator (requester) or responder (listener) for a GRASP negotiation session. After initiation, negotiation is a symmetric process, so most of the functions can be used by either party.

* Synchronization. These functions allow an ASA to to act as an initiator (requester) or responder (listener and data source) for a GRASP synchronization session.

* Flooding. These functions allow an ASA to send and receive an Objective that is flooded to all nodes of the ACP.

Some example logic flows for a resource management ASA are given in [I-D.ietf-anima-asa-guidelines], which may be of help in understanding the following descriptions. The next section describes parameters and data structures used in multiple API calls. The following sections describe various groups of function APIs. Those APIs that do not list asynchronous mechanisms are implicitly synchronous in their behaviour.

### 2.3.2. Parameters and data structures

#### 2.3.2.1. Errorcode

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

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

2. No reply: used in non-blocking calls to indicate that the other end has sent no reply so far (see Section 2.2).

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

Appendix A gives a full list of currently suggested error codes, based on implementation experience. While there is no absolute requirement for all implementations to use the same error codes, this is highly recommended for portability of applications.

#### 2.3.2.2. Timeout

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

#### 2.3.2.3. Objective

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

* name (UTF-8 string) - the objective's name

* neg (Boolean flag) - True if objective supports negotiation (default False)

* synch (Boolean flag) - True if objective supports synchronization (default False)

* dry (Boolean flag) - True if objective supports dry-run negotiation (default False)

  - Note 1: Only one of 'synch' or 'neg' may be True.

  - Note 2: 'dry' must not be True unless 'neg' is also True.

  - Note 3: In a language such as C the preferred implementation may be to represent the Boolean flags as bits in a single byte, which is how they are encoded in GRASP messages. In other languages an enumeration might be preferable.

* loop_count (integer) - Limit on negotiation steps etc. (default GRASP_DEF_LOOPCT, see [I-D.ietf-anima-grasp])

* value - a specific data structure expressing the value of the objective. The format is language dependent, with the constraint that it can be validly represented in CBOR.

  An essential requirement for all language mappings and all implementations is that, regardless of what other options exist for a language-specific representation of the value, there is always an option to use a raw CBOR data item as the value. The API will then wrap this with CBOR Tag 24 as an encoded CBOR data item [RFC7049] for transmission via GRASP, and unwrap it after reception.

  The 'name' and 'value' fields are of variable length. GRASP does not set a maximum length for these fields, but only for the total length of a GRASP message. Implementations might impose length limits.

  An example data structure definition for an objective in the C language, assuming the use of a particular CBOR library, is:

```
 typedef struct {
    char *name;
    uint8_t flags;            // flag bits as defined by GRASP
    int loop_count;
    int value_size;           // size of value in bytes
    cbor_mutable_data cbor_value;
                              // CBOR bytestring (libcbor/cbor/data.h)
                } objective;
```

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

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

#### 2.3.2.4. ASA_locator

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

* locator - The actual locator, either an IP address or an ASCII string.

* ifi (integer) - The interface identifier index via which this was discovered - probably no use to a normal ASA

* expire (system dependent type) - The time on the local system clock when this locator will expire from the cache

* The following cover all locator types currently supported by GRASP:

  - is_ipaddress (Boolean) - True if the locator is an IP address

  - is_fqdn (Boolean) - True if the locator is an FQDN

  - is_uri (Boolean) - True if the locator is a URI

  - Note: Depending on the programming language, these could be represented as a bit pattern or an enumeration.

* diverted (Boolean) - True if the locator was discovered via a Divert option

* protocol (integer) - Applicable transport protocol (IPPROTO_TCP or IPPROTO_UDP)

* port (integer) - Applicable port number

The 'locator' field is of variable length in the case of an FQDN or a URI. GRASP does not set a maximum length for this field, but only for the total length of a GRASP message. Implementations might impose length limits.

It should be noted that when one ASA discovers the ASA_locator of another, there is no explicit authentication mechanism. In accordance with the trust model provided by the secure ACP, ASAs are presumed to provide correct locators in response to discovery.

#### 2.3.2.5. Tagged_objective

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

* objective - An objective

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

#### 2.3.2.6. Asa_nonce

Although an authentication and authorization scheme for ASAs has not been defined, the API provides a very simple hook for such a scheme. When an ASA starts up, it registers itself with the GRASP core, which provides it with an opaque nonce that, although not cryptographically protected, would be difficult for a third party to predict. The ASA must present this nonce in future calls. This mechanism will prevent some elementary errors or trivial attacks such as an ASA manipulating an objective it has not registered to use.

Thus, in most calls, an 'asa_nonce' parameter is required. It is generated when an ASA first registers with GRASP, and the ASA must then store the asa_nonce and use it in every subsequent GRASP call. Any call in which an invalid nonce is presented will fail. It is an up to 32-bit opaque value (for example represented as a uint32_t, depending on the language). Since it is only used locally, not in GRASP messages, it is only required to be unique within the local GRASP instance. It is valid until the ASA terminates. It should be unpredictable; a possible implementation is to use the same mechanism that GRASP uses to generate Session Identifiers (see Section 2.3.2.7). Another possible implementation is to hash the name of the ASA with a locally defined secret key.

2.3.2.7. Session_nonce

In some calls, a 'session_nonce' parameter is required. This is an opaque data structure as far as the ASA is concerned, used to identify calls to the API as belonging to a specific GRASP session (see Section 2.2). The section 'Session Identifier' of [I-D.ietf-anima-grasp] explains how uniqueness of Session Identifiers is provided across the autonomic network. In fully threaded implementations this parameter might not be needed, but it is included to act as a session handle if necessary. It will also allow GRASP to detect and ignore malicious calls or calls from timed-out sessions. A likely implementation is to form the nonce from the underlying GRASP Session ID and the source address of the session.

2.3.3. Registration

These functions are used to register an ASA, and the objectives that it modifies, with the GRASP module. In the absence of an authorization model, these functions are very simple but they will avoid multiple ASAs choosing the same name, and will prevent multiple ASAs manipulating the same objective. If an authorization model is added to GRASP, these API calls would need to be modified accordingly.

* register_asa()

  All ASAs must use this call.

  - Input parameter:

      name of the ASA (UTF-8 string)

  - Return parameters:

      errorcode (integer)

      asa_nonce (integer) (if successful)

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

* deregister_asa()

  - Input parameters:

    asa_nonce (integer)

    name of the ASA (UTF-8 string)

  - Return parameter:

    errorcode (integer)

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

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

* register_objective()

  ASAs must use this call for any objective whose value they need to transmit by negotiation, synchronization or flooding.

  - Input parameters:

    asa_nonce (integer)

    objective (structure)

    ttl (integer - default GRASP_DEF_TIMEOUT)

    discoverable (Boolean - default False)

    overlap (Boolean - default False)

    local (Boolean - default False)

  - Return parameter:

    errorcode (integer)

  - This registers an objective that this ASA may modify and transmit to other ASAs. It is not necessary to register an objective that is only received by GRASP synchronization or flooding. The 'objective' becomes a candidate for discovery. However, discovery responses should not be enabled until the

ASA calls listen_negotiate() or listen_synchronize(), showing that it is able to act as a responder. The ASA may negotiate the objective or send synchronization or flood data. Registration is not needed for "read-only" operations, i.e., the ASA only wants to receive synchronization or flooded data for the objective concerned.

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

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

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

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

- This call may be repeated for multiple objectives.

* deregister_objective()

  - Input parameters:

    asa_nonce (integer)

    objective (structure)

  - Return parameter:

    errorcode (integer)

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

#### 2.3.4. Discovery

* discover()

  This function may be used by any ASA to discover peers handling a given objective.

- Input parameters:

  asa_nonce (integer)

  objective (structure)

  timeout (integer)

  minimum_TTL (integer)

- Return parameters:

  errorcode (integer)

  locator_list (structure)

- This returns a list of discovered 'ASA_locator's for the given objective. Note that this structure includes all the fields described in Section 2.3.2.4.

- If the parameter 'minimum_TTL' is greater than zero, any locally cached locators for the objective whose remaining time to live in milliseconds is less than or equal to 'minimum_TTL' are deleted first. Thus 'minimum_TTL' = 0 will flush all entries.

- If the parameter 'timeout' is zero, any remaining locally cached locators for the objective are returned immediately and no other action is taken. (Thus, a call with 'minimum_TTL' and 'timeout' both equal to zero is pointless.)

- If the parameter 'timeout' is greater than zero, GRASP discovery is performed, and all results obtained before the timeout in milliseconds expires are returned. If no results are obtained, an empty list is returned after the timeout. That is not an error condition.

- Asynchronous Mechanisms:

  o Threaded implementation: This should be called in a separate thread if asynchronous operation is required.

  o Event loop implementation: An additional read/write 'session_nonce' parameter is used. A callback may be used in the case of a non-zero tiemout.

### 2.3.5. Negotiation

Since the negotiation mechanism is different from a typical client/server exchange, Figure 2 illustrates the sequence of calls and GRASP messages in a negotiation. Note that after the first protocol exchange, the process is symmetrical and either side can end the negotiation. Similarly, either side can insert a delay at any time, to extend the other side's timeout.

```
Initiator                         Responder
---------                         ---------

                                  listen_negotiate() \ Await request

request_negotiate()
          M_REQ_NEG      ->       negotiate_step()   \ Open session,
                         <-      M_NEGOTIATE         / start negotiation
negotiate_step()
        M_NEGOTIATE      ->       negotiate_step()   \ Continue
                         <-      M_NEGOTIATE         / negotiation
                         ...
negotiate_wait()                                     \ Insert
        M_WAIT           ->                          / delay
negotiate_step()
        M_NEGOTIATE      ->       negotiate_step()   \ Continue
                         <-      M_NEGOTIATE         / negotiation
negotiate_step()
        M_NEGOTIATE      ->       end_negotiate()    \ End
                         <-      M_END               / negotiation

                                                     \ Process results
```

Figure 2: Negotiation sequence

* request_negotiate()

  This function is used by any ASA to initiate negotiation of a GRASP Objective as a requester (client).

  - Input parameters:

    asa_nonce (integer)

    objective (structure)

    peer (ASA_locator)

    timeout (integer)

- Return parameters:

  errorcode (integer)

  session_nonce (structure) (if successful)

  proffered_objective (structure) (if successful)

  reason (string) (if negotiation declined)

- This function opens a negotiation session between two ASAs. Note that GRASP currently does not support multi-party negotiation, which would need to be added as an extended function.

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

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

- The 'peer' parameter is the target node; it must be an 'ASA_locator' as returned by discover(). If 'peer' is null, GRASP discovery is automatically performed first to find a suitable peer (i.e., any node that supports the objective in question).

- The 'timeout' parameter is described in Section 2.3.2.2.

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

  1. The 'session_nonce' parameter is null. In this case the negotiation has succeeded in one step and the peer has accepted the request. The returned 'proffered_objective' contains the value accepted by the peer, which is therefore equal to the value in the requested 'objective'. For this reason, no session nonce is needed, since the session has ended.

  2. The 'session_nonce' parameter is not null. In this case negotiation must continue. The 'session_nonce' must be presented in all subsequent negotiation steps. The returned 'proffered_objective' contains the first value proffered by the negotiation peer. The contents of this instance of the objective must be used to prepare the next

negotiation step (see negotiate_step() below) because it contains the updated loop count, sent by the negotiation peer. The GRASP code automatically decrements the loop count by 1 at each step, and returns an error if it becomes zero.

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

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

- Asynchronous Mechanisms:

  o Threaded implementation: This should be called in a separate thread if asynchronous operation is required.

  o Event loop implementation: The 'session_nonce' parameter is used to distinguish multiple simultaneous sessions.

- Use of dry run mode: This must be consistent within a GRASP session. The state of the 'dry' flag in the initial request_negotiate() call must be the same in all subsequent negotiation steps of the same session. The semantics of the dry run mode are built into the ASA; GRASP merely carries the flag bit.

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

* listen_negotiate()

  This function is used by an ASA to start acting as a negotiation responder (listener) for a given GRASP objective.

  - Input parameters:

asa_nonce (integer)

objective (structure)

- Return parameters:

  errorcode (integer)

  session_nonce (structure) (if successful)

  requested_objective (structure) (if successful)

- This function instructs GRASP to listen for negotiation requests for the given 'objective'. It also enables discovery responses for the objective, as mentioned under register_objective() in Section 2.3.3.

- Asynchronous Mechanisms:

  o Threaded implementation: It will block waiting for an incoming request, so should be called in a separate thread if asynchronous operation is required. Unless there is an unexpected failure, this call only returns after an incoming negotiation request. If the ASA supports multiple simultaneous transactions, a new sub-thread must be spawned for each new session, so that listen_negotiate() can be called again immediately.

  o Event loop implementation: A 'session_nonce' parameter is used to distinguish individual sessions. If the ASA supports multiple simultaneous transactions, a new event must be inserted in the event loop for each new session, so that listen_negotiate() can be reactivated immediately.

- This call only returns (threaded model) or triggers (event loop) after an incoming negotiation request. When this occurs, 'requested_objective' contains the first value requested by the negotiation peer. The contents of this instance of the objective must be used in the subsequent negotiation call because it contains the loop count sent by the negotiation peer. The 'session_nonce' must be presented in all subsequent negotiation steps.

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

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

* stop_listen_negotiate()

  This function is used by an ASA to stop acting as a responder (listener) for a given GRASP objective.

  - Input parameters:

    asa_nonce (integer)

    objective (structure)

  - Return parameter:

    errorcode (integer)

  - Instructs GRASP to stop listening for negotiation requests for the given objective, i.e., cancels 'listen_negotiate'.

  - Asynchronous Mechanisms:

    o Threaded implementation: Must be called from a different thread than 'listen_negotiate'.

    o Event loop implementation: no special considerations.

* negotiate_step()

  This function is used by either ASA in a negotiation session to make the next step in negotiation.

  - Input parameters:

    asa_nonce (integer)

    session_nonce (structure)

    objective (structure)

    timeout (integer) as described in Section 2.3.2.2

- Return parameters:

  Exactly as for 'request_negotiate'

- Executes the next negotation step with the peer. The 'objective' parameter contains the next value being proffered by the ASA in this step. It must also contain the latest 'loop_count' value received from request_negotiate() or negotiate_step().

- Asynchronous Mechanisms:

  o Threaded implementation: Called in the same thread as the preceding 'request_negotiate' or 'listen_negotiate', with the same value of 'session_nonce'.

  o Event loop implementation: Must use the same value of 'session_nonce' returned by the preceding 'request_negotiate' or 'listen_negotiate'.

* negotiate_wait()

  This function is used by either ASA in a negotiation session to delay the next step in negotiation.

  - Input parameters:

    asa_nonce (integer)

    session_nonce (structure)

    timeout (integer)

  - Return parameters:

    errorcode (integer)

  - Requests the remote peer to delay the negotiation session by 'timeout' milliseconds, thereby extending the original timeout. This function simply triggers a GRASP Confirm Waiting message (see [I-D.ietf-anima-grasp] for details).

  - Asynchronous Mechanisms:

    o Threaded implementation: Called in the same thread as the preceding 'request_negotiate' or 'listen_negotiate', with the same value of 'session_nonce'.

o Event loop implementation: Must use the same value of 'session_nonce' returned by the preceding 'request_negotiate' or 'listen_negotiate'.

* end_negotiate()

This function is used by either ASA in a negotiation session to end a negotiation.

- Input parameters:

  asa_nonce (integer)

  session_nonce (structure)

  result (Boolean)

  reason (UTF-8 string)

- Return parameters:

  errorcode (integer)

- End the negotiation session.

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

  'reason' = optional string describing reason for decline.

- Asynchronous Mechanisms:

  o Threaded implementation: Called in the same thread as the preceding 'request_negotiate' or 'listen_negotiate', with the same value of 'session_nonce'.

  o Event loop implementation: Must use the same value of 'session_nonce' returned by the preceding 'request_negotiate' or 'listen_negotiate'.

### 2.3.6. Synchronization and Flooding

* synchronize()

This function is used by any ASA to cause synchronization of a GRASP Objective as a requester (client).

- Input parameters:

asa_nonce (integer)

objective (structure)

peer (ASA_locator)

timeout (integer)

- Return parameters:

  errorcode (integer)

  result (structure) (if successful)

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

- Since this is essentially a read operation, any ASA can do it, unless an authorization model is added to GRASP in future. Therefore the API checks that the ASA is registered, but the objective does not need to be registered by the calling ASA.

- If the 'peer' parameter is null, and the objective is already available in the local cache, the flooded objective is returned immediately in the 'result' parameter. In this case, the 'timeout' is ignored.

- Otherwise, synchronization with a discovered ASA is performed. If successful, the retrieved objective is returned in the 'result' parameter.

- The 'peer' parameter is an 'ASA_locator' as returned by discover(). If 'peer' is null, GRASP discovery is automatically performed first to find a suitable peer (i.e., any node that supports the objective in question).

- The 'timeout' parameter is described in Section 2.3.2.2.

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

- Asynchronous Mechanisms:

  - o Threaded implementation: Call in a separate thread if asynchronous operation is required.

  - o Event loop implementation: An additional read/write 'session_nonce' parameter is used.

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

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

* listen_synchronize()

  This function is used by an ASA to start acting as a synchronization responder (listener) for a given GRASP objective.

  - Input parameters:

    asa_nonce (integer)

    objective (structure)

  - Return parameters:

    errorcode (integer)

  - This instructs GRASP to listen for synchronization requests for the given objective, and to respond with the value given in the 'objective' parameter. It also enables discovery responses for the objective, as mentioned under register_objective() in Section 2.3.3.

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

* stop_listen_synchronize()

  This function is used by an ASA to stop acting as a synchronization responder (listener) for a given GRASP objective.

  - Input parameters:

    asa_nonce (integer)

    objective (structure)

  - Return parameters:

    errorcode (integer)

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

* flood()

  This function is used by an ASA to flood one or more GRASP objectives throughout the autonomic network.

  Note that each GRASP node caches all flooded objectives that it receive, until each one's time-to-live expires. Cached objectives are tagged with their origin as well as an expiry time, so multiple copies of the same objective may be cached simultaneously. Further details are given in the section 'Flood Synchronization Message' of [I-D.ietf-anima-grasp]

  - Input parameters:

    asa_nonce (integer)

    ttl (integer)

    tagged_objective_list (structure)

  - Return parameters:

    errorcode (integer)

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

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

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

  - The function checks that the ASA registered each objective.

- This call may be repeated whenever any value changes.

* get_flood()

  This function is used by any ASA to obtain the current value of a flooded GRASP objective.

  - Input parameters:

    asa_nonce (integer)

    objective (structure)

  - Return parameters:

    errorcode (integer)

    tagged_objective_list (structure) (if successful)

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

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

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

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

* expire_flood()

  This function may be used by an ASA to expire specific entries in the local GRASP flood cache.

  - Input parameters:

    asa_nonce (integer)

    tagged_objective (structure)

- Return parameters:

  errorcode (integer)

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

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

### 2.3.7. Invalid Message Function

* send_invalid()

  This function may be used by any ASA to stop an ongoing GRASP session.

  - Input parameters:

    asa_nonce (integer)

    session_nonce (structure)

    info (bytes)

  - Return parameters:

    errorcode (integer)

  - Sends a GRASP Invalid Message (M_INVALID) message, as described in [I-D.ietf-anima-grasp]. Should not be used if end_negotiate() would be sufficient. Note that this message may be used in response to any unicast GRASP message that the receiver cannot interpret correctly. In most cases this message will be generated internally by a GRASP implementation.

    'info' = optional diagnostic data. May be raw bytes from the invalid message.

## 3. Implementation Status [RFC Editor: please remove]

A prototype open source Python implementation of GRASP, including an API similar to this document, has been used to verify the concepts for the threaded model. It may be found at https://github.com/becarpenter/graspy with associated documentation and demonstration ASAs.

## 4. Security Considerations

Security considerations for the GRASP protocol are discussed in [I-D.ietf-anima-grasp]. These include denial of service issues, even though these are considered a low risk in the ACP. In various places GRASP recommends an exponential backoff. An ASA using the API should use exponential backoff after failed discover(), req_negotiate() or synchronize() operations. The timescale for such backoffs depends on the semantics of the GRASP objective concerned. Additionally, a flood() operation should not be repeated at shorter intervals than is useful. The appropriate interval depends on the semantics of the GRASP objective concerned. These precautions are intended to assist the detection of malicious denial of service attacks.

As a general precaution, all ASAs able to handle multiple negotiation or synchronization requests in parallel may protect themselves against a denial of service attack by limiting the number of requests they can handle simultaneously and silently discarding excess requests.

As noted earlier, the trust model is that all ASAs in a given autonomic network communicate via a secure autonomic control plane and therefore trust each other's messages. Specific authorization of ASAs to use particular GRASP objectives is a subject for future study, also briefly discussed in [I-D.ietf-anima-grasp].

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

## 5. IANA Considerations

This document makes no request of the IANA.

## 6. Acknowledgements

Excellent suggestions were made by Ignas Bagdonas, Laurent Ciavaglia, Toerless Eckert, Paul Kyzivat, Guangpeng Li, Michael Richardson, Joseph Salowey, Rob Wilton, and other participants in the ANIMA WG.

## 7. References

### 7.1. Normative References

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", Work in Progress, Internet-Draft, draft-ietf-anima-grasp-15, 13 July 2017, <https://tools.ietf.org/html/draft-ietf-anima-grasp-15>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

### 7.2. Informative References

[I-D.ciavaglia-anima-coordination]
Ciavaglia, L. and P. Peloso, "Autonomic Functions Coordination", Work in Progress, Internet-Draft, draft-ciavaglia-anima-coordination-01, 21 March 2016, <https://tools.ietf.org/html/draft-ciavaglia-anima-coordination-01>.

[I-D.ietf-anima-asa-guidelines]
Carpenter, B., Ciavaglia, L., Jiang, S., and P. Pierre, "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>.

[I-D.ietf-anima-autonomic-control-plane]
Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic Control Plane (ACP)", Work in Progress, Internet-Draft, draft-ietf-anima-autonomic-control-plane-30, 30 October 2020, <https://tools.ietf.org/html/draft-ietf-anima-autonomic-control-plane-30>.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-bootstrapping-keyinfra-45, 11 November 2020, <https://tools.ietf.org/html/draft-ietf-anima-bootstrapping-keyinfra-45>.

[I-D.ietf-anima-grasp-distribution]
Liu, B., Xiao, X., Hecker, A., Jiang, S., Despotovic, Z., and B. Carpenter, "Information Distribution over GRASP", Work in Progress, Internet-Draft, draft-ietf-anima-grasp-distribution-01, 1 September 2020, <https://tools.ietf.org/html/draft-ietf-anima-grasp-distribution-01>.

[I-D.ietf-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", Work in Progress, Internet-Draft, draft-ietf-anima-reference-model-10, 22 November 2018, <https://tools.ietf.org/html/draft-ietf-anima-reference-model-10>.

## Appendix A. Error Codes

This Appendix lists the error codes defined so far on the basis of implementation experience, with suggested symbolic names and corresponding descriptive strings in English. It is expected that complete API implementations will provide for localisation of these descriptive strings, and that additional error codes will be needed according to implementation details.

The error codes that may only be returned by one or two functions are annotated accordingly, and the others are more general. The 'noSecurity' error will be returned to most calls if GRASP is running in an insecure mode (no ACP), except for the specific DULL usage mode described in the section 'Discovery Unsolicited Link-Local' of [I-D.ietf-anima-grasp].

```
ok               0 "OK"
declined         1 "Declined" (req_negotiate, negotiate_step)
noReply          2 "No reply"
unspec           3 "Unspecified error"
ASAfull          4 "ASA registry full"  (register_asa)
dupASA           5 "Duplicate ASA name" (register_asa)
noASA            6 "ASA not registered"
notYourASA       7 "ASA registered but not by you"
                                        (deregister_asa)
notBoth          8 "Objective cannot support both negotiation
                    and synchronization" (register_obj)
notDry           9 "Dry-run allowed only with negotiation"
                                        (register_obj)
notOverlap      10 "Overlap not supported by this implementation"
                                        (register_obj)
objFull         11 "Objective registry full"
                                        (register_obj)
objReg          12 "Objective already registered"
                                        (register_obj)
notYourObj      13 "Objective not registered by this ASA"
notObj          14 "Objective not found"
notNeg          15 "Objective not negotiable"
                          (req_negotiate, listen_negotiate)
noSecurity      16 "No security"
```

```
   noDiscReply     17 "No reply to discovery"
                               (req_negotiate)
   sockErrNegRq    18 "Socket error sending negotiation request"
                               (req_negotiate)
   noSession       19 "No session"
   noSocket        20 "No socket"
   loopExhausted   21 "Loop count exhausted" (negotiate_step)
   sockErrNegStep  22 "Socket error sending negotiation step"
                                           (negotiate_step)
   noPeer          23 "No negotiation peer"
                               (req_negotiate, negotiate_step)
   CBORfail        24 "CBOR decode failure"
   invalidNeg      25 "Invalid Negotiate message"
                               (req_negotiate, negotiate_step)
   invalidEnd      26 "Invalid end message"
                               (req_negotiate, negotiate_step)
   noNegReply      27 "No reply to negotiation step"
                               (req_negotiate, negotiate_step)
   noValidStep     28 "No valid reply to negotiation step"
                               (req_negotiate, negotiate_step)
   sockErrWait     29 "Socket error sending wait message"
                               (negotiate_wait)
   sockErrEnd      30 "Socket error sending end message"
                               (end_negotiate, send_invalid)
   IDclash         31 "Incoming request Session ID clash"
                               (listen_negotiate)
   notSynch        32 "Not a synchronization objective"
                               (synchronize, get_flood)
   notFloodDisc    33 "Not flooded and no reply to discovery"
                               (synchronize)
   sockErrSynRq    34 "Socket error sending synch request"
                               (synchronize)
   noListener      35 "No synch listener"
                               (synchronize)
   noSynchReply    36 "No reply to synchronization request"
                               (synchronize)
   noValidSynch    37 "No valid reply to synchronization request"
                               (synchronize)
   invalidLoc      38 "Invalid locator" (flood)
```

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

draft-ietf-anima-grasp-api-08, 2020-11:

* Clarified trust model

* Added explanations of GRASP objectives and sessions

* Added note about non-idempotent messages

* Added overview of API functions, and annotated each function with a brief description

* Added protocol diagram for negotiation session

* Clarified (absence of) authorization model

* Changed precise semantics of synchronize() for flooded objectives

* Clarified caching of flooded objectives

* Changed 'age_limit' to 'minimum_TTL'

* Improved security considerations, including DOS precautions

* Annotated error codes to indicate which functions generate which errors

* Other clarifications from Last Call reviews

draft-ietf-anima-grasp-api-07, 2020-10-13:

* Improved diagram and its description

* Added pointer to example logic flows

* Added note on variable length parameters

* Clarified that API decrements loop count automatically

* Other corrections and clarifications from AD review

draft-ietf-anima-grasp-api-06, 2020-06-07:

* Improved diagram

* Numerous clarifications and layout changes

draft-ietf-anima-grasp-api-05, 2020-05-08:

* Converted to xml2rfc v3

* Editorial fixes.

draft-ietf-anima-grasp-api-04, 2019-10-07:

* Improved discussion of layering, mentioned daemon.

* Added callbacks and improved description of asynchronous operations.

* Described use case for 'session_nonce'.

* More explanation of 'asa_nonce'.

* Change 'discover' to use 'age_limit' instead of 'flush'.

* Clarified use of 'dry run'.

* Editorial improvements.

draft-ietf-anima-grasp-api-03, 2019-01-21:

* Replaced empty "logic flows" section by "implementation status".

* Minor clarifications.

* Editorial improvements.

draft-ietf-anima-grasp-api-02, 2018-06-30:

* Additional suggestion for event-loop API.

* Discussion of error code values.

draft-ietf-anima-grasp-api-01, 2018-03-03:

* Editorial updates

draft-ietf-anima-grasp-api-00, 2017-12-23:

* WG adoption

* Editorial improvements.

draft-liu-anima-grasp-api-06, 2017-11-24:

* Improved description of event-loop model.

* Changed intended status to Informational.

* Editorial improvements.

draft-liu-anima-grasp-api-05, 2017-10-02:

* Added send_invalid()

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

* Noted that simple nodes might not include the API.

* Minor clarifications.

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

* Changed error return to integers.

* Required all implementations to accept objective values in CBOR.

* Added non-blocking alternatives.

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

* Updated for draft-ietf-anima-grasp-09

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

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

* Editorial corrections

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

* Updated for draft-ietf-anima-grasp-05

* Editorial corrections

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

* Initial version

## Authors' Addresses

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

Email: brian.e.carpenter@gmail.com

Bing Liu (editor)
Huawei Technologies
Q14, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing
100095
P.R. China

Email: leo.liubing@huawei.com

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

Email: wdwang@bupt.edu.cn

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

Email: xygong@bupt.edu.cn

ANIMA M. Behringer, Ed.
Internet-Draft
Intended status: Informational B. Carpenter
Expires: May 27, 2019 Univ. of Auckland
T. Eckert
Futurewei Technologies Inc.
L. Ciavaglia
Nokia
J. Nobre
University of Vale do Rio dos Sinos
November 23, 2018

# A Reference Model for Autonomic Networking
draft-ietf-anima-reference-model-10

## Abstract

This document describes a reference model for Autonomic Networking for managed networks. It defines the behaviour of an autonomic node, how the various elements in an autonomic context work together, and how autonomic services can use the infrastructure.

## Status of This Memo



## Copyright Notice

Table of Contents

## 1. Introduction

The document "Autonomic Networking - Definitions and Design Goals" [RFC7575] explains the fundamental concepts behind Autonomic Networking, and defines the relevant terms in this space, as well as a high level reference model. [RFC7576] provides a gap analysis between traditional and autonomic approaches.

This document defines this reference model with more detail, to allow for functional and protocol specifications to be developed in an architecturally consistent, non-overlapping manner.

As discussed in [RFC7575], the goal of this work is not to focus exclusively on fully autonomic nodes or networks. In reality, most networks will run with some autonomic functions, while the rest of the network is traditionally managed. This reference model allows for this hybrid approach.

For example, it is possible in an existing, non-autonomic network to enrol devices in a traditional way, to bring up a trust infrastructure with certificates. This trust infrastructure could then be used to automatically bring up an Autonomic Control Plane (ACP), and run traditional network operations over the secure and self-healing ACP. See [I-D.ietf-anima-stable-connectivity] for a description of this use case.

The scope of this model is therefore limited to networks that are to some extent managed by skilled human operators, loosely referred to as "professionally managed" networks. Unmanaged networks raise additional security and trust issues that this model does not cover.

This document describes a first, simple, implementable phase of an Autonomic Networking solution. It is expected that the experience from this phase will be used in defining updated and extended specifications over time. Some topics are considered architecturally in this document, but are not yet reflected in the implementation specifications. They are marked with an (*).

## 2. The Network View

This section describes the various elements in a network with autonomic functions, and how these entities work together, on a high level. Subsequent sections explain the detailed inside view for each of the autonomic network elements, as well as the network functions (or interfaces) between those elements.

Figure 1 shows the high level view of an Autonomic Network. It consists of a number of autonomic nodes, which interact directly with each other. Those autonomic nodes provide a common set of capabilities across the network, called the "Autonomic Networking Infrastructure" (ANI). The ANI provides functions like naming, addressing, negotiation, synchronization, discovery and messaging.

Autonomic functions typically span several, possibly all nodes in the network. The atomic entities of an autonomic function are called the "Autonomic Service Agents" (ASA), which are instantiated on nodes.

```
   +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
   :            :       Autonomic Function 1        :                 :
   : ASA 1      :      ASA 1      :      ASA 1      :          ASA 1  :
   +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
                :                 :                 :
                :   +- - - - - - - - - - - - - - +  :
                :   :   Autonomic Function 2     :  :
                :   :  ASA 2      :      ASA 2   :  :
                :   +- - - - - - - - - - - - - - +  :
                :                 :                 :
   +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
   :                Autonomic Networking Infrastructure               :
   +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
   +--------+   :    +--------+   :    +--------+   :        +--------+
   | Node 1 |--------| Node 2 |--------| Node 3 |----...-----| Node n |
   +--------+   :    +--------+   :    +--------+   :        +--------+
```

Figure 1: High level view of an Autonomic Network

In a horizontal view, autonomic functions span across the network, as well as the Autonomic Networking Infrastructure. In a vertical view, a node always implements the ANI, plus it may have one or several Autonomic Service Agents. ASAs may be standalone, or use other ASAs in a hierarchical way.

The Autonomic Networking Infrastructure (ANI) therefore is the foundation for autonomic functions.

## 3. The Autonomic Network Element

This section explains the general architecture of an Autonomic Network Element (Section 3.1), how it tracks its surrounding environment in an Adjacency Table (Section 3.2), and the state machine which defines the behaviour of the network element (Section 3.3), based on that adjacency table.

### 3.1. Architecture

This section describes an autonomic network element and its internal architecture. The reference model explained in the document "Autonomic Networking - Definitions and Design Goals" [RFC7575] shows the sources of information that an autonomic service agent can leverage: Self-knowledge, network knowledge (through discovery), Intent (see Section 7.2), and feedback loops. There are two levels inside an autonomic node: the level of Autonomic Service Agents, and the level of the Autonomic Networking Infrastructure, with the former using the services of the latter. Figure 2 illustrates this concept.

```
+------------------------------------------------------------+
|                                                            |
| +-----------+        +------------+        +------------+  |
| | Autonomic |        | Autonomic  |        | Autonomic  |  |
| | Service   |        | Service    |        | Service    |  |
| | Agent 1   |        | Agent 2    |        | Agent 3    |  |
| +-----------+        +------------+        +------------+  |
|       ^                    ^                     ^         |
| -  -  | -  - API level -  -| -  -  -  -  -  -  - |-  -  -  |
|       V                    V                     V         |
|------------------------------------------------------------|
| Autonomic Networking Infrastructure                        |
|    - Data structures (ex: certificates, peer information)  |
|    - Generalized Autonomic Control Plane (GACP)            |
|    - Autonomic Node Addressing and naming                  |
|    - Discovery, negotiation and synchronisation functions  |
|    - Distribution of Intent and other information          |
|    - Aggregated reporting and feedback loops               |
|    - Routing                                               |
|------------------------------------------------------------|
|             Basic Operating System Functions               |
+------------------------------------------------------------+
```

Figure 2: Model of an autonomic node

The Autonomic Networking Infrastructure (lower part of Figure 2) contains node specific data structures, for example trust information about itself and its peers, as well as a generic set of functions, independent of a particular usage. This infrastructure should be generic, and support a variety of Autonomic Service Agents (upper part of Figure 2). It contains addressing and naming of autonomic nodes, discovery, negotiation and synchronisation functions, distribution of information, reporting and feedback loops, as well as routing inside the Autonomic Control Plane.

The Generalized Autonomic Control Plane (GACP) is the summary of all interactions of the Autonomic Networking Infrastructure with other nodes and services. A specific implementation of the GACP is referred to here as the Autonomic Control Plane (ACP), and described in [I-D.ietf-anima-autonomic-control-plane].

The use cases of "Autonomics" such as self-management, self-optimisation, etc, are implemented as Autonomic Service Agents. They use the services and data structures of the underlying Autonomic Networking Infrastructure, which should be self-managing.

The "Basic Operating System Functions" include the "normal OS", including the network stack, security functions, etc.

Full AN nodes have the full Autonomic Networking Infrastructure, with the full functionality described in this document. At a later stage ANIMA may define a scope for constrained nodes with a reduced ANI and well-defined minimal functionality. They are currently out of scope.

## 3.2. The Adjacency Table

Autonomic Networking is based on direct interactions between devices of a domain. The Autonomic Control Plane (ACP) is normally constructed on a hop-by-hop basis. Therefore, many interactions in the ANI are based on the ANI adjacency table. There are interactions that provide input into the adjacency table, and other interactions that leverage the information contained in it.

The ANI adjacency table contains information about adjacent autonomic nodes, at a minimum: node-ID, IP address in data plane, IP address in ACP, domain, certificate. An autonomic node maintains this adjacency table up to date. The adjacency table only contains information about other nodes that are capable of Autonomic Networking; non-autonomic nodes are normally not tracked here. However, the information is tracked independently of the status of the peer nodes; specifically, it contains information about non-enrolled nodes, nodes of the same and other domains. The adjacency table may contain information about the validity and trust level of the adjacent autonomic nodes.

The adjacency table is fed by the following inputs:

- o Link local discovery: This interaction happens in the data plane, using IPv6 link local addressing only, because this addressing type is itself autonomic. This way the nodes learns about all autonomic nodes around itself. The related standards track documents ([I-D.ietf-anima-grasp], [I-D.ietf-anima-bootstrapping-keyinfra], [I-D.ietf-anima-autonomic-control-plane]) describe in detail how link local discovery is used.

- o Vendor re-direct: A new device may receive information on where its home network is through a vendor based Manufacturer Authorized Signing Authority (MASA, see Section 5.3) re-direct; this is typically a routable address.

- o Non-autonomic input: A node may be configured manually with an autonomic peer; it could learn about autonomic nodes through DHCP options, DNS, and other non-autonomic mechanisms. Generally such non-autonomic mechansims require some administrator intervention. The key purpose is to by-pass a non-autonomic device or network. As this pertains to new devices, it is covered in appendix A and B of [I-D.ietf-anima-bootstrapping-keyinfra].

The adjacency table is defining the behaviour of an autonomic node:

- o If the node has not bootstrapped into a domain (i.e., doesn't have a domain certificate), it rotates through all nodes in the adjacency table that claim to have a domain, and will attempt bootstrapping through them, one by one. One possible response is a re-direct via a vendor MASA, which will be entered into the adjacency table (see second bullet above). See [I-D.ietf-anima-bootstrapping-keyinfra] for details.

- o If the adjacent node has the same domain, it will authenticate that adjacent node and, if successful, establish the Autonomic Control Plane (ACP). See [I-D.ietf-anima-autonomic-control-plane].

- o Once the node is part of the ACP of a domain, it will use GRASP [I-D.ietf-anima-grasp] to find Registrar(s) of its domain and potentially other services.

- o If the node is part of an ACP and has discovered at least one Registrar in its domain via GRASP, it will start the "join assistant" ASA, and act as a join assistant for neighboring nodes that need to be bootstrapped. See Section 6.3.1.2 for details.

- o Other behaviours are possible, for example establishing the ACP also with devices of a sub-domain, to other domains, etc. Those will likely be controlled by Intent. They are outside scope for the moment. Note that Intent is distributed through the ACP; therefore, a node can only adapt Intent driven behaviour once it has joined the ACP. At the moment, ANIMA does not consider providing Intent outside the ACP; this can be considered later.

Once a node has joined the ACP, it will also learn the ACP addresses of its adjacent nodes, and add them to the adjacency table, to allow for communication inside the ACP. Further autonomic domain interactions will now happen inside the ACP. At this moment, only negotiation / synchronization via GRASP [I-D.ietf-anima-grasp] is being defined. (Note that GRASP runs in the data plane, as an input in building the adjacency table, as well as inside the ACP.)

Autonomic Functions consist of Autonomic Service Agents (ASAs). They run logically above the AN Infrastructure, and may use the adjacency table, the ACP, negotiation and synchronization through GRASP in the ACP, Intent and other functions of the ANI. Since the ANI only provides autonomic interactions within a domain, autonomic functions can also use any other context on a node, specifically the global data plane.

### 3.3. State Machine

Autonomic Networking applies during the full life-cycle of a node. This section describes a state machine of an autonomic node, throughout its life.

A device is normally expected to store its domain specific identity, the LDevID (see Section 5.2), in persistent storage, to be available after a powercycle event. For device types that cannot store the LDevID in persistent storage, a powercycle event is effectively equivalent to a factory reset.

#### 3.3.1. State 1: Factory Default

An autonomic node leaves the factory in this state. In this state, the node has no domain specific configuration, specifically no LDevID, and could be used in any particular target network. It does however have a vendor/manufacturer specific ID, the IDevID [IDevID]. Nodes without IDevID cannot be autonomically and securely enrolled into a domain; they require manual pre-staging, in which case the pre-staging takes them directly to state 2.

Transitions:

- o Bootstrap event: The device enrols into a domain; as part of this process it receives a domain identity (LDevID). If enrollment is successful, the next state is state 2. See [I-D.ietf-anima-bootstrapping-keyinfra] Section 3 for details on enrollment.

- o Powercycle event: The device loses all state tables. It remains in state: 1.

#### 3.3.2. State 2: Enrolled

An autonomic node is in the state "enrolled" if it has a domain identity (LDevID), and has currently no ACP channel up. It may have further configuration or state, for example if it had been in state 3 before, but lost all its ACP channels. The LDevID can only be removed from a device through a factory reset, which also removes all other state from the device. This ensures that a device has no stale domain specific state when entering the "enrolled" state from state 1.

Transitions:

- o Joining ACP: The device establishes an ACP channel to an adjacent device. See [I-D.ietf-anima-autonomic-control-plane] for details. Next state: 3.

- o Factory reset: A factory reset removes all configuration and the domain identity (LDevID) from the device. Next state: 1.

- o Powercycle event: The device loses all state tables, but not its domain identity (LDevID). it remains in state: 2.

#### 3.3.3. State 3: In ACP

In this state, the autonomic node has at least one ACP channel to another device. The node can now participate in further autonomic transactions, such as starting autonomic service agents (e.g., it must now enable the join assistant ASA, to help other devices to join the domain. Other conditions may apply to such interactions, for example to serve as a join assistant, the device must first discover a bootstrap Registrar.

Transitions:

- o Leaving ACP: The device drops the last (or only) ACP channel to an adjacent device. Next state: 2.

o Factory reset: A factory reset removes all configuration and the domain identity (LDevID) from the device. Next state: 1.

o Powercycle event: The device loses all state tables, but not its domain identity (LDevID). Next state: 2.

## 4. The Autonomic Networking Infrastructure

The Autonomic Networking Infrastructure provides a layer of common functionality across an Autonomic Network. It provides the elementary functions and services, as well as extensions. An Autonomic Function, comprising of Autonomic Service Agents on nodes, uses the functions described in this section.

### 4.1. Naming

Inside a domain, each autonomic device should be assigned a unique name. The naming scheme should be consistent within a domain. Names are typically assigned by a Registrar at bootstrap time and persistent over the lifetime of the device. All Registrars in a domain must follow the same naming scheme.

In the absence of a domain specific naming scheme, a default naming scheme should use the same logic as the addressing scheme discussed in [I-D.ietf-anima-autonomic-control-plane]. The device name is then composed of a Registrar ID (for example taking a MAC address of the Registrar) and a device number. An example name would then look like this:

0123-4567-89ab-0001

The first three fields are the MAC address, the fourth field is the sequential number for the device.

### 4.2. Addressing

Autonomic Service Agents (ASAs) need to communicate with each other, using the autonomic addressing of the Autonomic Networking Infrastructure of the node they reside on. This section describes the addressing approach of the Autonomic Networking Infrastructure, used by ASAs.

Addressing approaches for the data plane of the network are outside the scope of this document. These addressing approaches may be configured and managed in the traditional way, or negotiated as a service of an ASA. One use case for such an autonomic function is described in [I-D.ietf-anima-prefix-management].

Autonomic addressing is a function of the Autonomic Networking Infrastructure (lower part of Figure 2), specifically the Autonomic Control Plane. ASAs do not have their own addresses. They may use either API calls, or the autonomic addressing scheme of the Autonomic Networking Infrastructure.

An autonomic addressing scheme has the following requirements:

- Zero-touch for simple networks: Simple networks should have complete self-management of addressing, and not require any central address management, tools, or address planning.

- Low-touch for complex networks: If complex networks require operator input for autonomic address management, it should be limited to high level guidance only, expressed in Intent.

- Flexibility: The addressing scheme must be flexible enough for nodes to be able to move around, for the network to grow, split and merge.

- Robustness: It should be as hard as possible for an administrator to negatively affect addressing (and thus connectivity) in the autonomic context.

- Stability: The addressing scheme should be as stable as possible. However, implementations need to be able to recover from unexpected address changes.

- Support for virtualization: Autonomic functions can exist either at the level of the physical network and physical devices, or at the level of virtual machines, containers and networks. In particular, Autonomic Nodes may support Autonomic Service Agents in virtual entities. The infrastructure, including the addressing scheme, should be able to support this architecture.

- Simplicity: To make engineering simpler, and to give the human administrator an easy way to trouble-shoot autonomic functions.

- Scale: The proposed scheme should work in any network of any size.

- Upgradability: The scheme must be able to support different addressing concepts in the future.

The proposed addressing scheme is described in the document "An Autonomic Control Plane" ([I-D.ietf-anima-autonomic-control-plane]).

## 4.3. Discovery

Traditionally, most of the information a node requires is provided through configuration or northbound interfaces. An autonomic function should rely on such northbound interfaces minimally or not at all, and therefore it needs to discover peers and other resources in the network. This section describes various discovery functions in an autonomic network.

Discovering nodes and their properties and capabilities: A core function to establish an autonomic domain is the mutual discovery of autonomic nodes, primarily adjacent nodes and secondarily off-link peers. This may in principle either leverage existing discovery mechanisms, or use new mechanisms tailored to the autonomic context. An important point is that discovery must work in a network with no predefined topology, ideally no manual configuration of any kind, and with nodes starting up from factory condition or after any form of failure or sudden topology change.

Discovering services: Network services such as AAA should also be discovered and not configured. Service discovery is required for such tasks. An autonomic network can either leverage existing service discovery functions, or use a new approach, or a mixture.

Thus the discovery mechanism could either be fully integrated with autonomic signaling (next section) or could use an independent discovery mechanism such as DNS Service Discovery or Service Location Protocol. This choice could be made independently for each Autonomic Service Agent, although the infrastructure might require some minimal lowest common denominator (e.g., for discovering the security bootstrap mechanism, or the source of information distribution, Section 4.7).

Phase 1 of Autonomic Networking uses GRASP for discovery, described in [I-D.ietf-anima-grasp].

## 4.4. Signaling Between Autonomic Nodes

Autonomic nodes must communicate with each other, for example to negotiate and/or synchronize technical objectives (i.e., network parameters) of any kind and complexity. This requires some form of signaling between autonomic nodes. Autonomic nodes implementing a specific use case might choose their own signaling protocol, as long as it fits the overall security model. However, in the general case, any pair of autonomic nodes might need to communicate, so there needs to be a generic protocol for this. A prerequisite for this is that autonomic nodes can discover each other without any preconfiguration, as mentioned above. To be generic, discovery and signaling must be able to handle any sort of technical objective, including ones that require complex data structures. The document "A Generic Autonomic Signaling Protocol (GRASP)" [I-D.ietf-anima-grasp] describes more detailed requirements for discovery, negotiation and synchronization in an autonomic network. It also defines a protocol, GRASP, for this purpose, including an integrated but optional discovery protocol.

GRASP is normally expected to run inside the Autonomic Control Plane (ACP; see Section 4.6) and to depend on the ACP for security. It may run insecurely for a short time during bootstrapping.

An autonomic node will normally run a single instance of GRASP, used by multiple ASAs. However, scenarios where multiple instances of GRASP run in a single node, perhaps with different security properties, are not excluded.

## 4.5. Routing

All autonomic nodes in a domain must be able to communicate with each other, and later phases also with autonomic nodes outside their own domain. Therefore, an Autonomic Control Plane relies on a routing function. For Autonomic Networks to be interoperable, they must all support one common routing protocol.

The routing protocol is defined in the ACP document [I-D.ietf-anima-autonomic-control-plane].

## 4.6. The Autonomic Control Plane

The "Autonomic Control Plane" carries the control protocols in an autonomic network. In the architecture described here, it is implemented as an overlay network. The document "An Autonomic Control Plane" ([I-D.ietf-anima-autonomic-control-plane]) describes the implementation details suggested here. This document uses the term "overlay" to mean a set of point-to-point adjacencies congruent with the underlying interconnection topology. The terminology may not be aligned with a common usage of the "overlay" term in routing context. See [I-D.ietf-anima-stable-connectivity] for uses cases for the ACP.

## 4.7. Information Distribution (*)

Certain forms of information require distribution across an autonomic domain. The distribution of information runs inside the Autonomic Control Plane. For example, Intent is distributed across an autonomic domain, as explained in [RFC7575].

Intent is the policy language of an Autonomic Network, see also Section 7.2. It is a high level policy, and should change only infrequently (order of days). Therefore, information such as Intent should be simply flooded to all nodes in an autonomic domain, and there is currently no perceived need to have more targeted distribution methods. Intent is also expected to be monolithic, and flooded as a whole. One possible method for distributing Intent, as well as other forms of data, is discussed in [I-D.liu-anima-grasp-distribution]. Intent and information distribution are not part of phase 1 of ANIMA.

## 5. Security and Trust Infrastructure

An Autonomic Network is self-protecting. All protocols are secure by default, without the requirement for the administrator to explicitly configure security, with the exception of setting up a PKI infrastructure.

Autonomic nodes have direct interactions between themselves, which must be secured. Since an autonomic network does not rely on configuration, it is not an option to configure, for example, pre-shared keys. A trust infrastructure such as a PKI infrastructure must be in place. This section describes the principles of this trust infrastructure. In this first phase of autonomic networking, a device is either within the trust domain and fully trusted, or outside the trust domain and fully untrusted.

The default method to automatically bring up a trust infrastructure is defined in the document "Bootstrapping Key Infrastructures" [I-D.ietf-anima-bootstrapping-keyinfra]. The ASAs required for this enrollment process are described in Section 6.3. An autonomic node must implement the enrollment and join assistant ASAs. The registrar ASA may be implemented only on a sub-set of nodes.

### 5.1. Public Key Infrastructure

An autonomic domain uses a PKI model. The root of trust is a certification authority (CA). A registrar acts as a registration authority (RA).

A minimum implementation of an autonomic domain contains one CA, one Registrar, and network elements.

### 5.2. Domain Certificate

Each device in an autonomic domain uses a domain certificate (LDevID) to prove its identity. A new device uses its manufacturer provided certificate (IDevID) during bootstrap, to obtain a domain certificate. [I-D.ietf-anima-bootstrapping-keyinfra] describes how a new device receives a domain certificate, and the certificate format.

## 5.3. The MASA

The Manufacturer Authorized Signing Authority (MASA) is a trusted service for bootstrapping devices. The purpose of the MASA is to provide ownership tracking of devices in a domain. The MASA provides audit, authorization, and ownership tokens to the registrar during the bootstrap process to assist in the authentication of devices attempting to join an Autonomic Domain, and to allow a joining device to validate whether it is joining the correct domain. The details for MASA service, security, and usage are defined in [I-D.ietf-anima-bootstrapping-keyinfra].

## 5.4. Sub-Domains (*)

By default, sub-domains are treated as different domains. This implies no trust between a domain and its sub-domains, and no trust between sub-domains of the same domain. Specifically, no ACP is built, and Intent is valid only for the domain it is defined for explicitly.

In phase 2 of ANIMA, alternative trust models should be defined, for example to allow full or limited trust between domain and sub-domain.

## 5.5. Cross-Domain Functionality (*)

By default, different domains do not interoperate, no ACP is built and no trust is implied between them.

In the future, models can be established where other domains can be trusted in full or for limited operations between the domains.

# 6. Autonomic Service Agents (ASA)

This section describes how autonomic services run on top of the Autonomic Networking Infrastructure.

## 6.1. General Description of an ASA

An Autonomic Service Agent (ASA) is defined in [RFC7575] as "An agent implemented on an autonomic node that implements an autonomic function, either in part (in the case of a distributed function) or whole." Thus it is a process that makes use of the features provided by the ANI to achieve its own goals, usually including interaction with other ASAs via the GRASP protocol [I-D.ietf-anima-grasp] or otherwise. Of course it also interacts with the specific targets of its function, using any suitable mechanism. Unless its function is very simple, the ASA will need to handle overlapping asynchronous operations. It may therefore be a quite complex piece of software in its own right, forming part of the application layer above the ANI. ASA design guidelines are available in [I-D.carpenter-anima-asa-guidelines].

Thus we can distinguish at least three classes of ASAs:

- Simple ASAs with a small footprint that could run anywhere.

- Complex, possibly multi-threaded ASAs that have a significant resource requirement and will only run on selected nodes.

- A few 'infrastructure ASAs' that use basic ANI features in support of the ANI itself, which must run in all autonomic nodes. These are outlined in the following sections.

Autonomic nodes, and therefore their ASAs, know their own capabilities and restrictions, derived from hardware, firmware or pre-installed software: They are "self-aware".

The role of an autonomic node depends on Intent and on the surrounding network behaviors, which may include forwarding behaviors, aggregation properties, topology location, bandwidth, tunnel or translation properties, etc. For example, a node may decide to act as a backup node for a neighbor, if its capabilities allow it to do so.

Following an initial discovery phase, the node properties and those of its neighbors are the foundation of the behavior of a specific node. A node and its ASAs have no pre-configuration for the particular network in which they are installed.

Since all ASAs will interact with the ANI, they will depend on appropriate application programming interfaces (APIs). It is desirable that ASAs are portable between operating systems, so these APIs need to be universal. An API for GRASP is described in [I-D.ietf-anima-grasp-api].

ASAs will in general be designed and coded by experts in a particular technology and use case, not by experts in the ANI and its components. Also, they may be coded in a variety of programming languages, in particular including languages that support object constructs as well as traditional variables and structures. The APIs should be designed with these factors in mind.

It must be possible to run ASAs as non-privileged (user space) processes except for those (such as the infrastructure ASAs) that necessarily require kernel privilege. Also, it is highly desirable that ASAs can be dynamically loaded on a running node.

Since autonomic systems must be self-repairing, it is of great importance that ASAs are coded using robust programming techniques. All run-time error conditions must be caught, leading to suitable minimally disruptive recovery actions, also considering a complete restart of the ASA. Conditions such as discovery failures or negotiation failures must be treated as routine, with the ASA retrying the failed operation, preferably with an exponential back-off in the case of persistent errors. When multiple threads are started within an ASA, these threads must be monitored for failures and hangups, and appropriate action taken. Attention must be given to garbage collection, so that ASAs never run out of resources. There is assumed to be no human operator - again, in the worst case, every ASA must be capable of restarting itself.

ASAs will automatically benefit from the security provided by the ANI, and specifically by the ACP and by GRASP. However, beyond that, they are responsible for their own security, especially when communicating with the specific targets of their function. Therefore, the design of an ASA must include a security analysis beyond 'use ANI security.'

### 6.2. ASA Life-Cycle Management

ASAs operating on a given ANI may come from different providers and pursue different objectives. Management of ASAs and its interactions with the ANI should follow the same operating principles, hence comply to a generic life-cycle management model.

The ASA life-cycle provides standard processes to:

- install ASA: copy the ASA code onto the node and start it,
- deploy ASA: associate the ASA instance with a (some) managed network device(s) (or network function),
- control ASA execution: when and how an ASA executes its control loop.

The life-cyle will cover the sequential states below: Installation, Deployment, Operation and the transitional states in-between. This Life-Cycle will also define which interactions ASAs have with the ANI in between the different states. The noticeable interactions are:

- o Self-description of ASA instances at the end of deployment: its format needs to define the information required for the management of ASAs by ANI entities

- o Control of ASA control-loop during the operation: a signaling has to carry formatted messages to control ASA execution (at least starting and stopping the control loop)

## 6.3. Specific ASAs for the Autonomic Network Infrastructure

The following functions provide essential, required functionality in an autonomic network, and are therefore mandatory to implement on unconstrained autonomic nodes. They are described here as ASAs that include the underlying infrastructure components, but implementation details might vary.

The first three together support the trust enrollment process described in Section 5. For details see [I-D.ietf-anima-bootstrapping-keyinfra].

### 6.3.1. The enrollment ASAs

#### 6.3.1.1. The Pledge ASA

This ASA includes the function of an autonomic node that bootstraps into the domain with the help of an join assitant ASA (see below). Such a node is known as a Pledge during the enrollment process. This ASA must be installed by default on all nodes that require an autonomic zero-touch bootstrap.

#### 6.3.1.2. The Join Assistant ASA

This ASA includes the function of an autonomic node that helps a non-enrolled, adjacent devices to enroll into the domain. This ASA must be installed on all nodes, although only one join assistant needs to be active on a given LAN. See also [I-D.ietf-anima-bootstrapping-keyinfra].

#### 6.3.1.3. The Join Registrar ASA

This ASA includes the join registrar function in an autonomic network. This ASA does not need to be installed on all nodes, but only on nodes that implement the Join Registrar function.

#### 6.3.2. The ACP ASA

This ASA includes the ACP function in an autonomic network. In particular it acts to discover other potential ACP nodes, and to support the establishment and teardown of ACP channels. This ASA must be installed on all nodes. For details see Section 4.6 and [I-D.ietf-anima-autonomic-control-plane].

#### 6.3.3. The Information Distribution ASA (*)

This ASA is currently out of scope in ANIMA, and provided here only as background information.

This ASA includes the information distribution function in an autonomic network. In particular it acts to announce the availability of Intent and other information to all other autonomic nodes. This ASA does not need to be installed on all nodes, but only on nodes that implement the information distribution function. For details see Section 4.7.

Note that information distribution can be implemented as a function in any ASA. See [I-D.liu-anima-grasp-distribution] for more details on how information is suggested to be distributed.

## 7. Management and Programmability

This section describes how an Autonomic Network is managed, and programmed.

### 7.1. Managing a (Partially) Autonomic Network

Autonomic management usually co-exists with traditional management methods in most networks. Thus, autonomic behavior will be defined for individual functions in most environments. Examples for overlap are:

- o Autonomic functions can use traditional methods and protocols (e.g., SNMP and NETCONF) to perform management tasks, inside and outside the ACP;
- o Autonomic functions can conflict with behavior enforced by the same traditional methods and protocols;
- o Traditional functions can use the ACP, for example if reachability on the data plane is not (yet) established.

The autonomic Intent is defined at a high level of abstraction. However, since it is necessary to address individual managed elements, autonomic management needs to communicate in lower-level interactions (e.g., commands and requests). For example, it is expected that the configuration of such elements be performed using NETCONF and YANG modules as well as the monitoring be executed through SNMP and MIBs.

Conflict can occur between autonomic default behavior, autonomic Intent, traditional management methods. Conflict resolution is achieved in autonomic management through prioritization [RFC7575]. The rationale is that manual and node-based management have a higher priority over autonomic management. Thus, the autonomic default behavior has the lowest priority, then comes the autonomic Intent (medium priority), and, finally, the highest priority is taken by node-specific network management methods, such as the use of command line interfaces.

## 7.2. Intent (*)

Intent is not covered in the current implementation specifications. This section discusses a topic for further research.

This section gives an overview of Intent, and how it is managed. Intent and Policy-Based Network Management (PBNM) is already described inside the IETF (e.g., PCIM) and in other SDOs (e.g., DMTF and TMF ZOOM).

Intent can be described as an abstract, declarative, high-level policy used to operate an autonomic domain, such as an enterprise network [RFC7575]. Intent should be limited to high level guidance only, thus it does not directly define a policy for every network element separately.

Intent can be refined to lower level policies using different approaches. This is expected in order to adapt the Intent to the capabilities of managed devices. Intent may contain role or function information, which can be translated to specific nodes [RFC7575]. One of the possible refinements of the Intent is using Event-Condition-Action (ECA) rules.

Different parameters may be configured for Intent. These parameters are usually provided by the human operator. Some of these parameters can influence the behavior of specific autonomic functions as well as the way the Intent is used to manage the autonomic domain.

Intent is discussed in more detail in [I-D.du-anima-an-intent]. Intent as well as other types of information are distributed via GRASP, see [I-D.liu-anima-grasp-distribution].

### 7.3. Aggregated Reporting (*)

Aggregated reporting is not covered in the current implementation specifications. This section discusses a topic for further research.

An Autonomic Network should minimize the need for human intervention. In terms of how the network should behave, this is done through an autonomic Intent provided by the human administrator. In an analogous manner, the reports which describe the operational status of the network should aggregate the information produced in different network elements in order to present the effectiveness of autonomic Intent enforcement. Therefore, reporting in an autonomic network should happen on a network-wide basis [RFC7575].

Multiple simultaneous events can occur in an autonomic network in the same way they can happen in a traditional network. However, when reporting to a human administrator, such events should be aggregated to avoid notifications about individual managed elements. In this context, algorithms may be used to determine what should be reported (e.g., filtering) and in which way and how different events are related to each other. Besides that, an event in an individual element can be compensated by changes in other elements to maintain a network-wide target which is described in the autonomic Intent.

Reporting in an autonomic network may be at the same abstraction level as Intent. In this context, the aggregated view of current operational status of an autonomic network can be used to switch to different management modes. Despite the fact that autonomic management should minimize the need for user intervention, possibly there are some events that need to be addressed by human administrator actions.

### 7.4. Feedback Loops to NOC (*)

Feedback loops are required in an autonomic network to allow the intervention of a human administrator or central control systems, while maintaining a default behaviour. Through a feedback loop an administrator must be prompted with a default action, and has the possibility to acknowledge or override the proposed default action.

Uni-directional notifications to the NOC, that do not propose any default action, and do not allow an override as part of the transaction are considered like traditional notification services, such as syslog. They are expected to co-exist with autonomic methods, but are not covered in this draft.

## 7.5. Control Loops (*)

Control loops are not covered in the current implementation specifications. This section discusses a topic for further research.

Control loops are used in autonomic networking to provide a generic mechanism to enable the Autonomic System to adapt (on its own) to various factors that can change the goals that the autonomic network is trying to achieve, or how those goals are achieved. For example, as user needs, business goals, and the ANI itself changes, self-adaptation enables the ANI to change the services and resources it makes available to adapt to these changes.

Control loops operate to continuously observe and collect data that enables the autonomic management system to understand changes to the behavior of the system being managed, and then provide actions to move the state of the system being managed toward a common goal. Self-adaptive systems move decision-making from static, pre-defined commands to dynamic processes computed at runtime.

Most autonomic systems use a closed control loop with feedback. Such control loops should be able to be dynamically changed at runtime to adapt to changing user needs, business goals, and changes in the ANI.

## 7.6. APIs (*)

APIs are not covered in the current implementation specifications. This section discusses a topic for further research.

Most APIs are static, meaning that they are pre-defined and represent an invariant mechanism for operating with data. An Autonomic Network should be able to use dynamic APIs in addition to static APIs.

A dynamic API is one that retrieves data using a generic mechanism, and then enables the client to navigate the retrieved data and operate on it. Such APIs typically use introspection and/or reflection. Introspection enables software to examine the type and properties of an object at runtime, while reflection enables a program to manipulate the attributes, methods, and/or metadata of an object.

APIs must be able to express and preserve the semantics of data models. For example, software contracts [Meyer97] are based on the principle that a software-intensive system, such as an Autonomic Network, is a set of communicating components whose interaction is based on precisely-defined specifications of the mutual obligations that interacting components must respect. This typically includes specifying:

- o pre-conditions that must be satisfied before the method can start execution

- o post-conditions that must be satisfied when the method has finished execution

- o invariant attributes that must not change during the execution of the method

## 7.7. Data Model (*)

Data models are not covered in the current implementation specifications. This section discusses a topic for further research.

The following definitions are adapted from [I-D.ietf-supa-generic-policy-data-model]:

An information model is a representation of concepts of interest to an environment in a form that is independent of data repository, data definition language, query language, implementation language, and protocol. In contrast, a data model is a representation of concepts of interest to an environment in a form that is dependent on data repository, data definition language, query language, implementation language, and protocol (typically, but not necessarily, all three).

The utility of an information model is to define objects and their relationships in a technology-neutral manner. This forms a consensual vocabulary that the ANI and ASAs can use. A data model is then a technology-specific mapping of all or part of the information model to be used by all or part of the system.

A system may have multiple data models. Operational Support Systems, for example, typically have multiple types of repositories, such as SQL and NoSQL, to take advantage of the different properties of each. If multiple data models are required by an Autonomic System, then an information model should be used to ensure that the concepts of each data model can be related to each other without technological bias.

A data model is essential for certain types of functions, such as a Model-Reference Adaptive Control Loop (MRACL). More generally, a data model can be used to define the objects, attributes, methods, and relationships of a software system (e.g., the ANI, an autonomic node, or an ASA). A data model can be used to help design an API, as well as any language used to interface to the Autonomic Network.

## 8. Coordination Between Autonomic Functions (*)

Coordination between autonomic functions is not covered in the current implementation specifications. This section discusses a topic for further research.

### 8.1. The Coordination Problem (*)

Different autonomic functions may conflict in setting certain parameters. For example, an energy efficiency function may want to shut down a redundant link, while a load balancing function would not want that to happen. The administrator must be able to understand and resolve such interactions, to steer autonomic network performance to a given (intended) operational point.

Several interaction types may exist among autonomic functions, for example:

- Cooperation: An autonomic function can improve the behavior or performance of another autonomic function, such as a traffic forecasting function used by a traffic allocation function.

- Dependency: An autonomic function cannot work without another one being present or accessible in the autonomic network.

- Conflict: A metric value conflict is a conflict where one metric is influenced by parameters of different autonomic functions. A parameter value conflict is a conflict where one parameter is modified by different autonomic functions.

Solving the coordination problem beyond one-by-one cases can rapidly become intractable for large networks. Specifying a common functional block on coordination is a first step to address the problem in a systemic way. The coordination life-cycle consists in three states:

- At build-time, a "static interaction map" can be constructed on the relationship of functions and attributes. This map can be used to (pre-)define policies and priorities on identified conflicts.

- At deploy-time, autonomic functions are not yet active/acting on the network. A "dynamic interaction map" is created for each instance of each autonomic functions and on a per resource basis, including the actions performed and their relationships. This map provides the basis to identify conflicts that will happen at run-time, categorize them and plan for the appropriate coordination strategies/mechanisms.

- o At run-time, when conflicts happen, arbitration is driven by the coordination strategies. Also new dependencies can be observed and inferred, resulting in an update of the dynamic interaction map and adaptation of the coordination strategies and mechanisms.

Multiple coordination strategies and mechanisms exist and can be devised. The set ranges from basic approaches such as random process or token-based process, to approaches based on time separation and hierarchical optimization, to more complex approaches such as multi-objective optimization, and other control theory approaches and algorithms family.

## 8.2. A Coordination Functional Block (*)

A common coordination functional block is a desirable component of the ANIMA reference model. It provides a means to ensure network properties and predictable performance or behavior such as stability, and convergence, in the presence of several interacting autonomic functions.

A common coordination function requires:

- o A common description of autonomic functions, their attributes and life-cycle.

- o A common representation of information and knowledge (e.g., interaction maps).

- o A common "control/command" interface between the coordination "agent" and the autonomic functions.

Guidelines, recommendations or BCPs can also be provided for aspects pertaining to the coordination strategies and mechanisms.

## 9. Security Considerations

In this section we distinguish outsider and insider attacks. In an outsider attack all network elements and protocols are securely managed and operating, and an outside attacker can sniff packets in transit, inject and replay packets. In an insider attack, the attacker has access to an autonomic node or other means (e.g. remote code execution in the node by exploiting ACP-independent vulnerabilities in the node platform) to produce arbitrary payloads on the protected ACP channels.

If a system has vulnerabilities in the implementation or operation (configuration), an outside attacker can exploit such vulnerabilies to become an insider attacker.

### 9.1. Protection Against Outsider Attacks

Here, we assume that all systems involved in an autonomic network are secured and operated according to best current practices. These protection methods comprise traditional security implementation and operation methods (such as code security, strong randomization algorithms, strong passwords, etc.) as well as mechanisms specific to an autonomic network (such as a secured MASA service).

Traditional security methods for both implementation and operation are outside scope for this document.

AN specific protocols and methods must also follow traditional security methods, in that all packets that can be sniffed or injected by an outside attacker are:

- protected against modification.
- authenticated.
- protected against replay attacks.
- confidentiality protected (encrypted).
- and that the AN protocols are robust against packet drops and man-in-the-middle attacks.

How these requirements are met is covered in the AN standards track documents that define the methods used, specifically [I-D.ietf-anima-bootstrapping-keyinfra], [I-D.ietf-anima-grasp], and [I-D.ietf-anima-autonomic-control-plane].

Most AN messages run inside the cryptographically protected ACP. The unprotected AN messages outside the ACP are limited to a simple discovery method, defined in Section 2.5.2 of [I-D.ietf-anima-grasp]: The "Discovery Unsolicited Link-Local (DULL)" message, with detailed rules on its usage.

If AN messages can be observed by a third party, they might reveal valuable information about network configuration, security precautions in use, individual users, and their traffic patterns. If encrypted, AN messages might still reveal some information via traffic analysis.

## 9.2. Risk of Insider Attacks

An autonomic network consists of autonomic devices that form a distributed self-managing system. Devices within a domain have credentials issued from a common trust anchor and can use them to create mutual trust. This means that any device inside a trust domain can by default use all distributed functions in the entire autonomic domain in a malicious way.

If an autonomic node or protocol has vulnerabilities or is not securely operated, an outside attacker has the following generic ways to take control of an autonomic network:

- Introducing a fake device into the trust domain, by subverting the authentication methods. This depends on the correct specification, implementation and operation of the AN protocols.

- Subverting a device which is already part of a trust domain, and modifying its behavior. This threat is not specific to the solution discussed in this document, and applies to all network solutions.

- Exploiting potentially yet unknown protocol vulnerabilities in the AN or other protocols. Also this is a generic threat that applies to all network solutions.

The above threats are in principle comparable to other solutions: In the presence of design, implementation or operational errors, security is no longer guaranteed. However, the distributed nature of AN, specifically the Autonomic Control Plane, increases the threat surface significantly. For example, a compromised device may have full IP reachability to all other devices inside the ACP, and can use all AN methods and protocols.

For the next phase of the ANIMA work it is therefore recommended to introduce a sub-domain security model, to reduce the attack surface and not expose a full domain to a potential intruder. Furthermore, additional security mechanisms on the ASA level should be considered for high-risk autonomic functions.

# 10. IANA Considerations

This document requests no action by IANA.

## 11. Acknowledgements

Many people have provided feedback and input to this document: Sheng Jiang, Roberta Maglione, Jonathan Hansford, Jason Coleman, Artur Hecker. Useful reviews were made by Joel Halpern, Radia Perlman, Tianran Zhou and Christian Hopps.

## 12. Contributors

Significant contributions to this document have been made by John Strassner and Bing Liu from Huawei, and Pierre Peloso from Nokia.

## 13. References

### 13.1. Normative References

[I-D.ietf-anima-autonomic-control-plane]
Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-ietf-anima-autonomic-control-plane-18 (work in progress), August 2018.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-17 (work in progress), November 2018.

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", draft-ietf-anima-grasp-15 (work in progress), July 2017.

[IDevID] IEEE Standard, , "IEEE 802.1AR Secure Device Identifier", December 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

### 13.2. Informative References

[I-D.carpenter-anima-asa-guidelines]
Carpenter, B., Ciavaglia, L., Jiang, S., and P. Pierre, "Guidelines for Autonomic Service Agents", draft-carpenter-anima-asa-guidelines-05 (work in progress), June 2018.

[I-D.du-anima-an-intent]
Du, Z., Jiang, S., Nobre, J., Ciavaglia, L., and M. Behringer, "ANIMA Intent Policy and Format", draft-du-anima-an-intent-05 (work in progress), February 2017.

[I-D.ietf-anima-grasp-api]
Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)", draft-ietf-anima-grasp-api-02 (work in progress), June 2018.

[I-D.ietf-anima-prefix-management]
Jiang, S., Du, Z., and B. Carpenter, "Autonomic IPv6 Edge Prefix Management in Large-scale Networks", draft-ietf-anima-prefix-management-07 (work in progress), December 2017.

[I-D.ietf-anima-stable-connectivity]
Eckert, T. and M. Behringer, "Using Autonomic Control Plane for Stable Connectivity of Network OAM", draft-ietf-anima-stable-connectivity-10 (work in progress), February 2018.

[I-D.ietf-supa-generic-policy-data-model]
Halpern, J. and J. Strassner, "Generic Policy Data Model for Simplified Use of Policy Abstractions (SUPA)", draft-ietf-supa-generic-policy-data-model-04 (work in progress), June 2017.

[I-D.liu-anima-grasp-distribution]
Liu, B., Jiang, S., Xiao, X., Hecker, A., and Z. Despotovic, "Information Distribution in Autonomic Networking", draft-liu-anima-grasp-distribution-09 (work in progress), October 2018.

[Meyer97] Meyer, B., "Object-Oriented Software Construction (2nd edition)", Prentice-Hall, ISBN 978-0136291558, 1997.

[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>.

## Authors' Addresses

Michael H. Behringer (editor)

Email: Michael.H.Behringer@gmail.com

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

Email: brian.e.carpenter@gmail.com

Toerless Eckert
Futurewei Technologies Inc.
2330 Central Expy
Santa Clara 95050
USA

Email: tte@cs.fau.de

Laurent Ciavaglia
Nokia
Villarceaux
Nozay 91460
FR

Email: laurent.ciavaglia@nokia.com

Jeferson Campos Nobre
University of Vale do Rio dos Sinos
Av. Unisinos, 950
Sao Leopoldo 91501-970
Brazil

Email: jcnobre@unisinos.br

ANIMA WG B. Liu
INTERNET-DRAFT Huawei Technologies
Intended Status: Standard Track X. Xiao
Expires: June 14, 2020 A. Hecker
MRC, Huawei Technologies
S. Jiang
Huawei Technologies
Z. Despotovic
MRC, Huawei Technologies
December 12, 2019

# Information Distribution in Autonomic Networking
draft-liu-anima-grasp-distribution-13

## Abstract

This document proposes a solution for information distribution in autonomic networks. Information distribution is categorized into two different modes: 1) instantaneous distribution; 2) publication for retrieval. In the former case, the information is sent, propagates and is disposed of after reception. In the latter case, information needs to be stored in the network.

The capabilities to distribute information are basic and fundamental needs for an autonomous network (cf. ANI [I-D.ietf-anima-reference-model]). This document describes typical use cases of information distribution in ANI and requirements to ANI, such that rich information distribution can be natively supported. The document proposes extensions to the autonomic nodes and suggests an implementation based on GRASP extensions as a protocol on the wire.

## Status of this Memo

http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

## Copyright and License Notice



## Table of Contents

## 1 Introduction

In an autonomic network, autonomic functions (AFs) running on autonomic nodes constantly exchange information, e.g. AF control/management signaling or AF data exchange. This document discusses the information distribution capability of such exchanges between AFs.

Depending on the number of participants, the information can be distributed in in the following scenarios:

1) Point-to-point (P2P) Communication: information is exchanged between parties, i.e. two nodes.

2) One-to-Many Communication: information exchanges involve an information source and multiple receivers.

The approaches to information distribution can be chiefly categorized into two basic modes:

1) An instantaneous mode (push): a source sends the actual content (e.g. control/management signaling, synchronization data and so on) to all interested receiver(s) immediately. Generally, some preconfiguration is required, as nodes interested in this information must be already known to all nodes in the sense that any receiving node must be able to decide, to which nodes this data is to be sent.

2) An asynchronous mode (delayed pull): here, a source publishes the content in some form in the network, which may later be looked for, found and retrieved by some endpoints in the AN. Here, depending on the size of the content, either the whole content or only its metadata might be published into the AN. In the latter case the metadata (e.g. a content descriptor, e.g. a key, and a location in the ANI) may be used for the actual retrieval. Importantly, the source, i.e. here publisher, needs to be able to determine the node, where the information (or its metadata) can be stored.

To avoid repetitive implementations by each AF developer, this document opts for a common support for information distribution implemented as a basic ANI capability, therefore available to all AFs. In fact, GRASP already provides part of the capabilities.

Regardless, an AF may still define and implement its own information distribution capability. Such a capability may then be advertised using the common information distribution capability defined in this document. Overall, ANI nodes and AFs may decide, which of the information distribution mechanisms they want to use for which type of information, according to their own preferences (e.g. semantic routing table, etc.)

This document first analyzes requirements for information distribution in autonomic networks (Section 3) and then discuss the relevant node behavior (Section 4). After that, the required GRASP extensions are formally introduced (Section 5).

## 2. Terminology

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

## 3. Requirements for Information Distribution in ANI

The question of information distribution in an autonomic network can be discussed through particular use cases or more generally. Depending on the situation it can be quite simple or might require more complex provisions.

Indeed, in the simplest case, the information can be sent:
1) at once (in one packet, in one flow),
2) straightaway (send-and-forget),
3) to all nodes.

Presuming 1), 2) and 3) hold, information distribution in smaller or scarce topologies can be implemented using broadcast, i.e. unconstrained flooding. For reasons well-understood, this approach has its limits in larger and denser networks. In this case, a graph can be constructed such that it contains every node exactly once (e.g. a spanning tree), still allowing to distribute any information to all nodes straightaway. Multicast tree construction protocols could be used in this case. There are reasonable use cases for such scenarios, as presented in Appendix B.

A more complex scenario arises, if only 1) and 2) hold, but the information only concerns a subset of nodes. Then, some kind of selection becomes required, to which nodes the given information should be distributed. Here, a further distinction is necessary; notably, if the selection of the target nodes is with respect to the nature or position of the node, or whether it is with respect to the information content. If the first, some knowledge about the node types, its topological position, etc (e.g. the routing information within ANI) can be used to distinguish nodes accordingly. For instance, edge nodes and forwarding nodes can be distinguished in this way. If the distribution scope is primarily to be defined by the information elements, then a registration / join / subscription or label distribution mechanism is unavoidable. This would be the case, for instance, if the AFs can be dynamically deployed on nodes, and the information is majorily destined to the AFs. Then, depending on the current AF deployment, the distribution scope must be adjusted as well.

If only 1) holds, but the information content might be required again and again, or might not yet be fully available, then more complex mechanisms might be required to store the information within the network for later, for further redistribution, and for notification of interested nodes. Examples for this include distribution of reconfiguration information for different AF instances, which might not require an immediate action, but only an eventual update of the parameters. Also, in some situations, there could be a significant delay between the occurrence of a new event and the full content availability (e.g. if the processing requires a lot of time).

Finally, none of the three might hold. Then, along with the subscription and notification, the actual content might be different from its metadata, i.e. some description of the content and, possibly, its location. The fetching can then be implemented in different, appropriate ways, if necessary as a complex transport session.

In essence, as flooding is usually not an option, and the interest of nodes for particular information elements can change over time, ANI should support autonomics also for the information distribution.

This calls for autonomic mechanisms in the ANI, allowing participating nodes to 1) advertise or publish 2) look for or subscribe to 3) store 4) fetch/retrieve 5) instantaneously push information elements.

In the following cases, situations depicting diverse information distribution needs are discussed.

1) Long Communication Intervals. The actual sending of the information is not necessarily instantaneous with some event. Advanced AFs may involve into longer jobs/tasks (e.g. database lookup, authentication etc.) when processing requests, and might not be able to reply immediately. Instead of actively waiting for the reply, a better way for an interested AF might be to get notified, when the reply is finally available.

2) Common Interest Distribution. AFs may share interest in common information. For example, the network intent will be distributed to network nodes enrolled, which is usually one-to-many scenario. Intent distribution can also be performed by an instant flooding (e.g. via GRASP) to every network node. However, because of network dynamics, not every node can be just ready at the moment when the network intent is broadcast. Also, a flooding often does not cover all network nodes as there is usually a limitation on the hop number. In fact, nodes may join in the network sequentially. In this situation, an asynchronous communication model could be a better choice where every (newly joining) node can subscribe the intent information and will get notified if it is ready (or updated).

3) Distributed Coordination. With computing and storage resources on autonomic nodes, alive AFs not only consume but also generate data information. An example is AFs coordinating with each other as distributed schedulers, responding to service requests and

distributing tasks. It is critical for those AFs to make correct decisions based on local information, which might be asymmetric as well. AFs may also need synthetic/aggregated data information (e.g. statistic info, like average values of several AFs, etc.) to make decisions. In these situations, AFs will need an efficient way to form a global view of the network (e.g. about resource consumption, bandwidth and statistics). Obviously, purely relying on instant communication model is inefficient, while a scalable, common, yet distributed data layer, on which AFs can store and share information in an asynchronous way, should be a better choice.

Therefore, for ANI, in order to support various communication scenarios, an information distribution module is required, and both instantaneous and asynchronous communication models should be supported. Some real-world use cases are introduced in Appendix A.

## 4. Node Behaviors

In this section, how a node should behave in order to support the two identified modes of information distribution is discussed. An ANI is a distributed system, so the information distribution module must be implemented in a distributed way as well.

### 4.1 Instant Information Distribution (IID) Sub-module

In this case, An information sender directly specifies the information receiver(s). The instant information distribution sub-module will be the main element.

#### 4.1.1 Instant P2P Communication

IID sub-module performs instant information transmission for ASAs running in an ANI. In specific, IID sub-module will have to retrieve the address of the information receiver specified by an ASA, then deliver the information to the receiver. Such a delivery can be done either in a connectionless or a connection-oriented way.

Current GRASP provides the capability to support instant P2P synchronization for ASAs. A P2P synchronization is a use case of P2P information transmission. However, as mention in Section 3, there are some scenarios where one node needs to transmit some information to another node(s). This is different to synchronization because after transmitting the information, the local status of the information does not have to be the same as the information sent to the receiver. This is not directly support by existing GRASP.

#### 4.1.2 Instant Flooding Communication

IID sub-module finishes instant flooding for ASAs in an ANI. Instant flooding is for all ASAs in an ANI. An information sender has to specify a special destination address of the information and broadcast to all interfaces to its neighbors. When another IID sub-module receives such a broadcast, after checking its TTL, it further broadcast the message to the neighbors. In order to avoid flooding storms in an ANI, usually a TTL number is specified, so that after a pre-defined limit, the flooding message will not be further broadcast again.

In order to avoid unnecessary flooding, a selective flooding can be done where an information sender wants to send information to multiple receivers at once. When doing this, sending information needs to contain criteria to judge on which interfaces the distributed information should and should not be sent. Specifically, the criteria contain:

o Matching Condition: a set of matching rules such as addresses of recipients, node features and so on.

o Action: what the node needs to do when the Matching Condition is fulfilled. For example, the action could be forwarding or discarding the distributed message.

Sent information must be included in the message distributed from the sender. The receiving node reacts by first checking the carried Matching Condition in the message to decide who should consume the message, which could be either the node itself, some neighbors or both. If the node itself is a recipient, Action field is followed; if a neighbor is a recipient, the message is sent accordingly.

An exemplary extension to support selective flooding on GRASP is described in Section 5.

### 4.2 Asynchronous Information Distribution (AID) Sub-module

In asynchronous information distribution, sender(s) and receiver(s) are not immediately specified while they may appear in an asynchronous way. Firstly, AID sub-module enables that the information can be stored in the network; secondly, AID sub-module provides an information publication and subscription (Pub/Sub) mechanism for ASAs.

As sketched in the previous section, in general each node requires two modules: 1) Information Storage (IS) module and 2) Event Queue (EQ) module in the information distribution module. Details of the two modules are described in the following sections.

### 4.2.1 Information Storage

IS module handles how to save and retrieve information for ASAs across the network. The IS module uses a syntax to index information, generating the hash index value (e.g. a hash value) of the information and mapping the hash index to a certain node in ANI. Note that, this mechanism can use existing solutions. Specifically, storing information in an ANIMA network will be realized in the following steps.

1) ASA-to-IS Negotiation. An ASA calls the API provided by information distribution module (directly supported by IS sub-module) to request to store the information somewhere in the network. The IS module performs various checks of the request (e.g. permitted information size).

2) Storing Peer Mapping. The information block will be handled by the IS module in order to calculate/map to a peer node in the network. Since ANIMA network is a peer-to-peer network, a typical way is to use distributed hash table (DHT) to map information to a unique index identifier. For example, if the size of the information is reasonable, the information block itself can be hashed, otherwise, some meta-data of the information block can be used to generate the mapping.

3) Storing Peer Negotiation Request. Negotiation request of storing the information will be sent from the IS module to the IS module on the destination node. The negotiation request contains parameters about the information block from the source IS module. According to the parameters as well as the local available resource, the requested storing peer will send feedback the source IS module.

4) Storing Peer Negotiation Response. Negotiation response from the storing peer is sent back to the source IS module. If the source IS module gets confirmation that the information can be stored, source IS module will prepare to transfer the information block; otherwise, a new storing peer must be discovered (i.e. going to step 7).

5) Information Block Transfer. Before sending the information block to the storing peer that already accepts the request, the IS module of the source node will check if the information block can be afforded by one GRASP message. If so, the information block will be directly sent by calling a GRASP API. Otherwise, a bulk data transmission is needed. For that, there are multiple ways to

do it.

The first option is to utilize one of existing protocols that is independent of the GRASP stack. For example, a session connectivity can be established to the storing peer, and over the connection the bulky data can be transmitted part by part. In this case, the IS module should support basic TCP-based session protocols such as HTTP(s) or native TCP.

The second option is to directly use GRASP itself for bulky data transferring. [I-D.carpenter-anima-grasp-bulk-04].

6) Information Writing. Once the information block (or a smaller block) is received, the IS module of the storing peer will store the data block in the local storage is accessible.

7) (Optional) New Storing Peer Discovery. If the previously selected storing peer is not available to store the information block, the source IS module will have to identify a new destination node to start a new negotiation. In this case, the discovery can be done by using discovery GRASP API to identify a new candidate, or more complex mechanisms can be introduced.

Similarly, Getting information from an ANI will be realized in the following steps.

1) ASA-to-IS Request. An ASA accesses the IS module via the APIs exposed by the information distribution module. The key/index of the interested information will be sent to the IS module. An assumption here is that the key/index should be known to an ASA before an ASA can ask for the information. This relates to the publishing/subscribing of the information, which are handled by other modules (e.g. Event Queue with Pub/Sub supported by GRASP).

2) Storing Peer Mapping. IS module maps the key/index of the requested information to a peer that stores the information, and prepares the information request. The mapping here follows the same mechanism when the information is stored.

3) Retrieval Negotiation Request. The source IS module sends a request to the storing peer and asks if such an information object is available.

4) Retrieval Negotiation Response. The storing peer checks the key/index of the information in the request, and replies to the source IS module. If the information is found and the information block can be afforded within one GRASP message, the information will be sent together with the response to the source IS module.

5) (Optional) New Destination Request. If the information is not found after the source IS module gets the response from the originally identified storing peer, the source IS module will have to discover the location of the requested information.

IS module can reuse distributed databases and key value stores like NoSQL, Cassandra, DHT technologies. storage and retrieval of information are all event-driven responsible by the EQ module.

4.2.2 Event Queue The Event Queue (EQ) module is to help ASAs to publish information to the network and subscribe to interested information in asynchronous scenarios. In an ANI, information generated on network nodes is an event labeled with an event ID, which is semantically related to the topic of the information. Key features of EQ module are summarized as follows.

1) Event Group: An EQ module provides isolated queues for different event groups. If two groups of AFs could have completely different purposes, the EQ module allows to create multiple queues where only AFs interested in the same topic will be aware of the corresponding event queue.

2) Event Prioritization: Events can have different priorities in ANI. This corresponds to how much important or urgent the event implies. Some of them are more urgent than regular ones. Prioritization allows AFs to differentiate events (i.e. information) they publish or subscribe to.

3) Event Matching: an information consumer has to be identified from the queue in order to deliver the information from the provider. Event matching keeps looking for the subscriptions in the queue to see if there is an exact published event there. Whenever a match is found, it will notify the upper layer to inform the corresponding ASAs who are the information provider and subscriber(s) respectively.

The EQ module on every network node operates as follows.

1) Event ID Generation: If information of an ASA is ready, an event ID is generated according to the content of the information. This is also related to how the information is stored/saved by the IS module introduced before. Meanwhile, the type of the event is also specified where it can be of control purpose or user plane data.

2) Priority Specification: According to the type of the event, the ASA may specify its priority to say how this event is to be processed. By considering both aspects, the priority of the event will be determined.

3) Event Enqueue: Given the event ID, event group and its priority, a queue is identified locally if all criteria can be satisfied. If there is such a queue, the event will be simply added into the queue, otherwise a new queue will be created to accommodate such an event.

4) Event Propagation: The published event will be propagated to the other network nodes in the ANIMA domain. A propagation algorithm can be employed to optimize the propagation efficiency of the updated event queue states.

5) Event Match and Notification: While propagating updated event states, EQ module in parallel keeps matching published events and its interested consumers. Once a match is found, the provider and subscriber(s) will be notified for final information retrieval.

The category of event priority is defined as the following. In general, there are two event types:

1) Network Control Event: This type of events are defined by the ANI for operational purposes on network control. A pre-defined priority levels for required system messages is suggested. For highest level to lowest level, the priority value ranges from NC_PRIOR_HIGH to NC_PRIOR_LOW as integer values. The NC_PRIOR_* values will be defined later according to the total number system events required by the ANI;

2) Custom ASA Event: This type of events are defined by the ASAs of users. This specifies the priority of the message within a group of ASAs, therefore it is only effective among ASAs that join the same message group. Within the message group, a group header/leader has to define a list of priority levels ranging from CUST_PRIOR_HIGH to CUST_PRIOR_LOW. Such a definition completely depends on the individual purposes of the message group.

When a system message is delivered, its event type and event priority value have to be both specified;

Event contains the address where the information is stored, after a subscriber is notified, it directly retrieves the information from the given location.

### 4.3 Summary

In summary, the general requirements for the information distribution module on each autonomic node are realized by two sub-modules handling instant communications and asynchronous communications, respectively. For instantaneous mode, node requirements are simple, calling for support for additional signaling. With minimum efforts, reusing the existing GRASP is possible.

For asynchronous mode, information distribution module uses new primitives on the wire, and implements an event queue and an information storage mechanism. An architectural consideration on ANI with the information distribution module is briefly discussed in Appendix B.

# 5. Extending GRASP for Information Distribution

## 5.1 Realizing Instant P2P Transmission

This could be a new message in GRASP. In fragmentary CDDL, an Un-solicited Synchronization message follows the pattern:

```
unsolicited_synch-message = [M_UNSOLIDSYNCH, session-id,
objective]
```

A node MAY actively send a unicast Un-solicited Synchronization message with the Synchronization data, to another node. This MAY be sent to port GRASP_LISTEN_PORT at the destination address, which might be obtained by GRASP Discovery or other possible ways. The synchronization data are in the form of GRASP Option(s) for specific synchronization objective(s).

## 5.2 Realizing Instant Selective Flooding

Since normal flooding is already supported by GRASP, this section only defines the selective flooding extension.

In fragmentary CDDL, the selective flooding follows the pattern:

```
   selective-flood-option = [O_SELECTIVE_FLOOD, +O_MATCH-CONDITION,
match-object, action]

      O_MATCH-CONDITION = [O_MATCH-CONDITION, Obj1, match-rule, Obj2]
      Obj1 = text

      match-rule = GREATER / LESS / WITHIN / CONTAIN

      Obj2 = text

      match-object = NEIGHBOR / SELF

      action = FORWARD / DROP
```

The option field encapsulates a match-condition option which represents the conditions regarding to continue or discontinue flood the current message. For the match-condition option, the Obj1 and Obj2 are to objects that need to be compared. For example, the Obj1 could be the role of the device and Obj2 could be "RSG". The match rules between the two objects could be greater, less than, within, or contain. The match-object represents of which Obj1 belongs to, it could be the device itself or the neighbor(s) intended to be flooded. The action means, when the match rule applies, the current device just continues flood or discontinues.

## 5.3 Realizing Subscription as An Event

In fragmentary CDDL, a Subscription Objective Option follows the pattern:

```
subscription-objection-option = [SUBSCRIPTION, 2, 2, subobj]
objective-name = SUBSCRIPTION

objective-flags = 2

loop-count = 2

subobj = text
```

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a subscription to a specific object.

## 5.4 Un_Subscription Objective Option

In fragmentary CDDL, a Un_Subscribe Objective Option follows the pattern:

```
Unsubscribe-objection-option = [UNSUBSCRIB, 2, 2, unsubobj]
objective-name = SUBSCRIPTION
objective-flags = 2
loop-count = 2
unsubobj = text
```

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a un-subscription to a specific object.

## 5.5 Publishing Objective Option

In fragmentary CDDL, a Publish Objective Option follows the pattern:

publish-objection-option = [PUBLISH, 2, 2, pubobj] objective-name
= PUBLISH
objective-flags = 2
loop-count = 2
pubobj = text

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a publish of a specific object data.

6. Security Considerations

The distribution source authentication could be done at multiple layers:

- o Outer layer authentication: the GRASP communication is within ACP (Autonomic Control Plane, [I-D.ietf-anima-autonomic-control-plane]). This is the default GRASP behavior.

- o Inner layer authentication: the GRASP communication might not be within a protected channel, then there should be embedded protection in distribution information itself. Public key infrastructure might be involved in this case.

7. IANA Considerations

TBD.

8. References

8.1 Normative References

[I-D.ietf-anima-grasp]
Bormann, D., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", draft-ietf-animagrasp-15 (Standard Track), October 2017.

8.2 Informative References

[RFC7575] Behringer, M., "Autonomic Networking: Definitions and Design Goals", RFC 7575, June 2015

[I-D.ietf-anima-autonomic-control-plane]
Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-behringer-anima-autonomic-control-plane-13, December 2017.

[I-D.ietf-anima-stable-connectivity-10]

Eckert, T., Behringer, M., "Using Autonomic Control Plane for Stable Connectivity of Network OAM", draft-ietf-anima-stable-connectivity-10, February 2018.

[I-D.ietf-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., Pierre P., Liu, B., Nobre, J., and J. Strassner, "A Reference Model for Autonomic Networking", draft-ietf-anima-reference-model-05, October 2017.

[I-D.du-anima-an-intent]
Du, Z., Jiang, S., Nobre, J., Ciavaglia, L., and M. Behringer, "ANIMA Intent Policy and Format", draft-duanima-an-intent-05 (work in progress), February 2017.

[I-D.ietf-anima-grasp-api]
Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)", draft-ietf-anima-grasp-api-00 (work in progress), December 2017.

[I-D.carpenter-anima-grasp-bulk-04]
Carpenter, B., Jiang, S., Liu, B., "Transferring Bulk Data over the GeneRic Autonomic Signaling Protocol (GRASP)", draft-carpenter-anima-grasp-bulk-04 (work in progress), July 3, 2019

[3GPP.29.500]
3GPP, "Technical Realization of Service Based Architecture", 3GPP TS 29.500 15.1.0, 09 2018

[3GPP.23.501]
3GPP, "System Architecture for the 5G System", 3GPP TS 23.501 15.2.0, 6 2018, <http://www.3gpp.org/ftp/Specs/html-info/23501.htm>.

[3GPP.23.502]
3GPP, "Procedures for the 5G System", 3GPP TS 23.502 15.2.0, 6 2018, <http://www.3gpp.org/ftp/Specs/html-info/23502.htm>.

[5GAA.use.cases]
White Paper "Toward fully connected vehicles: Edge computing for advanced automotive communications", 5GAA <http://5gaa.org/news/toward-fully-connected-vehicles-edge-computing-for-advanced-automotive-communications/>

Authors' Addresses

Bing Liu
Huawei Technologies
Q27, Huawei Campus

No.156 Beiqing Road
Hai-Dian District, Beijing  100095
P.R. China

Email: leo.liubing@huawei.com

Sheng Jiang
Huawei Technologies
Q27, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing  100095
P.R. China

Email: jiangsheng@huawei.com

Xun Xiao
Munich Research Center
Huawei technologies
Riesstr. 25, 80992, Muenchen, Germany

Emails: xun.xiao@huawei.com

Artur Hecker
Munich Research Center
Huawei technologies
Riesstr. 25, 80992, Muenchen, Germany

Emails: artur.hecker@huawei.com

Zoran Despotovic
Munich Research Center
Huawei technologies
Riesstr. 25, 80992, Muenchen, Germany

Emails: zoran.despotovic@huawei.com

Appendix A. Real-world Use Cases of Information Distribution

The requirement analysis in Section 3 shows that generally information distribution should be better of as an infrastructure layer module, which provides to upper layer utilizations. In this section, we review some use cases from the real-world where an information distribution module with powerful functions do plays a critical role there.

A.1 Service-Based Architecture (SBA) in 3GPP 5G

In addition to Internet, the telecommunication network (i.e. carrier mobile wireless networks) is another world-wide networking system. The architecture of the 5G mobile networks from 3GPP has been defined to follow a service-based architecture (SBA) where any network function (NF) can be dynamically associated with any other NF(s) when needed to compose a network service. Note that one NF can simultaneously associate with multiple other NFs, instead of being physically wired as in the previous generations of mobile networks. NFs communicate with each other over service-based interface (SBI), which is also standardized by 3GPP [3GPP.23.501].

In order to realize an SBA network system, detailed requirements are further defined to specify how NFs should interact with each other with information exchange over the SBI. We now list three requirements that are related to information distribution here.

1) NF Pub/Sub: Any NF should be able to expose its service status to the network and any NF should be able to subscribe the service status of an NF and get notified if the status is available. A concrete example is that a session management function (SMF) can subscribe to the REGISTER notification from an access management function (AMF) if there is a new user equipment trying to access the mobile network [3GPP.23.502].

2) Network Exposure Function (NEF): A particular network function that is required to manage the event exposure and distributions. Specifically, SBA requires such a functionality to register network events from the other NFs (e.g. AMF, SMF and so on), classify the events and properly handle event distributions accordingly in terms of different criteria (e.g. priorities) [3GPP.23.502].

3) Network Repository Function (NRF): A particular network function where all service status information is stored for the whole network. An SBA network system requires all NFs to be stateless so as to improve the resilience as well as agility of providing network services. Therefore, the information of the available NFs and the service status generated by those NFs will

be globally stored in NRF as a repository of the system. This clearly implies storage capability that keeps the information in the network and provides those information when needed. A concrete example is that whenever a new NF comes up, it first of all registers itself at NRF with its profile. When a network service requires a certain NF, it first inquires NRF to retrieve the availability information and decides whether or not there is an available NF or a new NF must be instantiated [3GPP.23.502].

(Note: 3GPP CT adopted HTTP2.0/JSON to be the protocol communicating between NFs, but autonomic networks can also load HTTP2.0 with in ACP.)

A.2 Vehicle-to-Everything

Connected car is one of scenarios interested in automotive manufacturers, carriers and vendors. 5G Automotive Alliance - an industry collaboration organization defines many promising use cases where services from car industry should be supported by the 5G mobile network. Here we list two examples as follows [5GAA.use.cases].

1) Software/Firmware Update: Car manufacturers expect that the software/firmware of their car products can be remotely updated/upgraded via 5G network, instead of onsite visiting their 4S stores/dealers offline as nowadays. This requires the network to provide a mechanism for vehicles to receive the latest software updates during a certain period of time. In order to run such a service for a car manufacturer, the network shall not be just like a network pipe anymore. Instead, information data have to be stored in the network, and delivered in a publishing/subscribing fashion. For example, the latest release of a software will be first distributed and stored at the access edges of the mobile network, after that, the updates can be pushed by the car manufacturer or pulled by the car owner as needed.

2) Real-time HD Maps: Autonomous driving clearly requires much finer details of road maps. Finer details not only include the details of just static road and streets, but also real-time information on the road as well as the driving area for both local urgent situations and intelligent driving scheduling. This asks for situational awareness at critical road segments in cases of changing road conditions. Clearly, a huge amount of traffic data that are real-time collected will have to be stored and shared across the network. This clearly requires the storage capability, data synchronization and event notifications in urgent cases from the network, which are still missing at the infrastructure layer.

A.3 Summary

Through the general analysis and the concrete examples from the real-world, we realize that the ways information are exchanged in the coming new scenarios are not just short and instant anymore. More advanced as well as diverse information distribution capabilities are required and should be generically supported from the infrastructure layer. Upper layer applications (e.g. ASAs in ANIMA) access and utilize such a unified mechanism for their own services.

## Appendix B. Information Distribution Module in ANI

This section describes how the information distribution module fits into the ANI and what extensions of GRASP are required [I-D.ietf-anima-grasp].

```
                +--------------------+
                |        ASAs        |
                +--------------------+
                        ^
                        |
                        v
    +-------------Info-Dist. APIs--------------+
    | +---------------+ +---------------------+ |
    | | Instant Dist. | | Asynchronous Dist.  | |
    | +---------------+ +---------------------+ |
    +------------------------------------------+
                        ^
                        |
                        v
                +---GRASP APIs----+
                |        ACP       |
                +-----------------+
```

Figure 1. Information Distribution Module and GRASP Extension.

As the Fig 1 shows, the information distribution module two sub-modules for instant and asynchronous information distributions, respectively, and provides APIs to ASAs. Specific Behaviors of modules are described in Section 5.

## Appendix C. Asynchronous ID Integrated with GRASP APIs

Actions triggered to the information distribution module will eventually invoke underlying GRASP APIs. Moreover, EQ and IS modules are usually correlated. When an AF(ASA) publishes information, not only such an event is translated and sent to EQ module, but also the information is indexed and stored simultaneously. Similarly, when an AF(ASA) subscribes information, not only subscribing event is triggered and sent to EQ module, but also the information will be retrieved by IS module at the same time.

o Storing and publishing information: This action involves both IS and EQ modules where a node that can store the information will be discovered first and related event will e published to the network. For this, GRASP APIs discover(), synchronize() and flood() are combined to compose such a procedure. In specific, discover() call will specific its objective being to "store_data" and the return parameters could be either an ASA_locator who will accept to store the data, or an error code indicating that no one could afford such data; after that, synchronize() call will send the data to the specified ASA_locator and the data will be stored at that node, with return of processing results like store_data_ack; meanwhile, such a successful event (i.e. data is stored successfully) will be flooded via a flood() call to interesting parties (such a multicast group existed).

o Subscribing and getting information: This action involves both IS and EQ modules as well where a node that is interested in a topic will subscribe the topic by triggering EQ module and if the topic is ready IS module will retrieve the content of the topic (i.e. the data). GRASP APIs such as register_objective(), flood(), synchronize() are combined to compose the procedure. In specific, any subscription action received by EQ module will be translated to register_objective() call where the interested topic will be the parameter inside of the call; the registration will be (selectively) flooded to the network by an API call of flood() with the option we extended in this draft; once a matched topic is found (because of the previous procedure), the node finding such a match will call API synchronize() to send the stored data to the subscriber.

6tisch Working Group M. Richardson
Internet-Draft Sandelman Software Works
Intended status: Informational P. van der Stok
Expires: August 18, 2018 vanderstok consultancy
P. Kampanakis
Cisco Systems
February 14, 2018

# Constrained Voucher Profile for Bootstrapping Protocols
draft-richardson-anima-ace-constrained-voucher-03

## Abstract

This document defines a strategy to securely assign a pledge to an owner, using an artifact signed, directly or indirectly, by the pledge's manufacturer. This artifact is known as a "voucher".

This document builds upon the work in [I-D.ietf-anima-voucher], encoding the resulting artifact in CBOR. Use with two signature technologies are described.

Additionally, this document explains how constrained vouchers may be transported in the [I-D.vanderstok-ace-coap-est] protocol.

## Status of This Memo



## Copyright Notice

Table of Contents

## 1. Introduction

Enrollment of new nodes into constrained networks with constrained nodes present unique challenges.

There are bandwidth and code space issues to contend. A solution such as [I-D.ietf-anima-bootstrapping-keyinfra] may be too large in terms of code space or bandwidth required.

This document defines a constrained version of [I-D.ietf-anima-voucher]. Rather than serializing the YANG definition in JSON, it is serialized into CBOR ([RFC7049]).

This document follows a similar, but not identical structure as [I-D.ietf-anima-voucher]. Some sections are left out entirely. Additional sections to [I-D.ietf-anima-voucher] concern: - Addition of voucher-request specification as defined in [I-D.ietf-anima-bootstrapping-keyinfra], - Addition to [I-D.vanderstok-ace-coap-est] of voucher transport requests over coap.

The CBOR definitions for this constrained voucher format are defined using the mechanism describe in [I-D.ietf-core-yang-cbor] using the SID mechanism explained in [I-D.ietf-core-sid]. As the tooling to convert YANG documents into an list of SID keys is still in its infancy, the table of SID values presented here should be considered normative rather than the output of the pyang tool.

Two methods of signing the resulting CBOR object are described in this document. One is CMS [RFC5652]. The other is COSE [RFC8152] signatures.

## 2. Terminology

The following terms are defined in [I-D.ietf-anima-voucher], and are used identically as in that document: artifact, imprint, domain, Join Registrar/Coordinator (JRC), Manufacturer Authorized Signing Authority (MASA), pledge, Trust of First Use (TOFU), and Voucher.

## 3. Requirements Language

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant STuPiD implementations.

## 4. Survey of Voucher Types

[I-D.ietf-anima-voucher] provides for vouchers that assert proximity, that authenticate the registrar and that include different amounts of anti-replay protection.

This document does not make any extensions to the types of vouchers.

Time based vouchers are included in this definition, but given that constrained devices are extremely unlikely to know the correct time, their use is very unlikely. Most users of these constrained vouchers will be online and will use live nonces to provide anti-replay protection.

[I-D.ietf-anima-voucher] defined only the voucher artifact, and not the Voucher Request artifact, which was defined in [I-D.ietf-anima-bootstrapping-keyinfra].

This document defines both a constrained voucher and a constrained voucher-request. They are presented in the order voucher-request, followed by voucher response as this is the time order that they occur.

## 5. Discovery and URI

This section describes the BRSKI extensions to EST-coaps [I-D.vanderstok-ace-coap-est] to transport the voucher between registrar, proxy and pledge over CoAP.

The extension is targeted to low-resource networks with small packets. Saving header space is important and the EST-coaps URI is shorter than the EST URI.

The presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "ace.est" [RFC6690]. Upon success, the return payload will contain the root resource of the EST resources. It is up to the implementation to choose its root resource; throughout this document the example root resource /est is used. The example below shows the discovery of the presence and location of voucher resources.

```
REQ: GET /.well-known/core?rt=ace.est

RES: 2.05 Content
</est>; rt="ace.est"
```

The EST-coaps server URIs differ from the EST URI by replacing the scheme https by coaps and by specifying shorter resource path names:

```
coaps://www.example.com/est/short-name
```

Figure 5 in section 3.2.2 of [RFC7030] enumerates the operations and corresponding paths which are supported by EST. Table 1 provides the mapping from the BRSKI extension URI path to the EST-coaps URI path.

| BRSKI | EST-coaps |
|---|---|
| /requestvoucher | /rv |
| /voucher-status | /vs |
| /enrollstatus | /es |
| /requestauditlog | /ra |

Table 1: BRSKI path to EST-coaps path

/requestvoucher and /enrollstatus are needed between pledge and Registrar.

When discovering the root path for the EST resources, the server MAY return the full resource paths and the used content types. This is useful when multiple content types are specified for EST-coaps server. For example, the following more complete response is possible.

```
REQ: GET /.well-known/core?rt=ace.est

RES: 2.05 Content
</est>; rt="ace.est"
</est/rv>; rt="ace.est";ct=50 TBD2 16
</est/vs>; rt="ace.est";ct=50
</est/es>; rt="ace.est";ct=50
</est/ra>; rt="ace.est";ct= TBD2 16
```

ct=50 stands for the Content-Format "application/json", ct=16 stands for the Content-Format "application/cose", and ct=TBD2 stands for Content-Format "application/voucher-cms+cbor defined in this document.

The return of the content-types allows the client to choose the most appropriate one from multiple content types.

# 6. Artifacts

This section describes the abstract (tree) definition as explained in [I-D.ietf-netmod-yang-tree-diagrams] first. This provides a high-level view of the contents of each artifact.

Then the assigned SID values are presented. These have been assigned using the rules in [I-D.ietf-core-yang-cbor], with an allocation that was made via the http://comi.space service.

((EDNOTE: it is unclear if there is further IANA work))

## 6.1. Voucher Request artifact

### 6.1.1. Tree Diagram

```
module: ietf-cwt-voucher-request

  grouping voucher-request-cwt-grouping
    +---- voucher
       +---- created-on
       |       yang:date-and-time
       +---- expires-on?
       |       yang:date-and-time
       +---- assertion
       |       enumeration
       +---- serial-number                                string
       +---- idevid-issuer?                               binary
       +---- pinned-domain-cert                           binary
       +---- domain-cert-revocation-checks?               boolean
       +---- nonce?                                       binary
       +---- last-renewal-date?
       |       yang:date-and-time
       +---- proximity-registrar-subject-public-key-info?   binary
```

### 6.1.2. SID values

[EDNote: the appropriate generation of the SID values is under discussion]

```
         SID Assigned to
   --------- --------------------------------------------------
     1001150 module ietf-cwt-voucher-request
     1001151 module ietf-restconf
     1001152 module ietf-voucher
     1001153 module ietf-yang-types
     1001154 data .../ietf-cwt-voucher-request:voucher
     1001155 data .../assertion
     1001156 data .../created-on
     1001157 data .../domain-cert-revocation-checks
     1001158 data .../expires-on
     1001159 data .../idevid-issuer
     1001160 data .../last-renewal-date
     1001161 data .../nonce
     1001162 data .../pinned-domain-cert
     1001163 data .../proximity-registrar-subject-public-key-info
     1001164 data .../serial-number
```

### 6.1.3. YANG Module

[EDNote: the appropriate syntax of the module is under discussion]

```
<CODE BEGINS> file "ietf-cwt-voucher-request@2017-12-11.yang"
/* -*- c -*- */
module ietf-cwt-voucher-request {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-cwt-voucher-request";
  prefix "vcwt";

  import ietf-voucher {
    prefix "v";
  }

  organization
   "IETF 6tisch Working Group";

  contact
   "WG Web:   <http://tools.ietf.org/wg/6tisch/>
    WG List:  <mailto:6tisch@ietf.org>
    Author:   Michael Richardson
              <mailto:mcr+ietf@sandelman.ca>";

  description
   "This module defines the format for a voucher, which is produced by
    a pledge’s manufacturer or delegate (MASA) to securely assign one
    or more pledges to an ’owner’, so that the pledges may establish a
```

```
    secure connection to the owner’s network infrastructure.

    This version provides a very restricted subset appropriate
    for very constrained devices.
    In particular, it assumes that nonce-ful operation is
    always required, that expiration dates are rather weak, as no
    clocks can be assumed, and that the Registrar is identified
    by a pinned Raw Public Key.

    The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT',
    'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in
    the module text are to be interpreted as described in RFC 2119.";

  revision "2017-12-11" {
    description
     "Initial version";
    reference
     "RFC XXXX: Voucher Profile for Constrained Devices";
  }

  // Grouping defined for future usage
  grouping voucher-request-cwt-grouping {
    description
      "Grouping to allow reuse/extensions in future work.";

    uses v:voucher-artifact-grouping {
      augment "voucher" {
        description "Base the CWT voucher-request upon the regular one";
        leaf proximity-registrar-subject-public-key-info {
          type binary;
          description
            "The proximity-registrar-subject-public-key-info replaces
         the proximit-registrar-cert in constrained uses of
         the voucher-request.
         The proximity-registrar-subject-public-key-info is the
         Raw Public Key of the Registrar. This field is encoded
         as specified in RFC7250, section 3.
         The ECDSA algorithm MUST be supported.
         The EdDSA algorithm as specified in
         draft-ietf-tls-rfc4492bis-17 SHOULD be supported.
         Support for the DSA algorithm is not recommended.
         Support for the RSA algorithm is a MAY.";
        }
      }
    }
  }
}
<CODE ENDS>
```

#### 6.1.4. Example voucher request artifacts

TBD

### 6.2. Voucher artifact

The voucher's primary purpose is to securely assign a pledge to an owner. The voucher informs the pledge which entity it should consider to be its owner.

This document defines a voucher that is a CBOR encoded instance of the YANG module defined in Section 5.3 that has been signed with CMS or with COSE.

### 6.3. Tree Diagram

```
module: ietf-cwt-voucher

  grouping voucher-cwt-grouping
    +---- voucher
       +---- created-on
       |       yang:date-and-time
       +---- expires-on?
       |       yang:date-and-time
       +---- assertion                               enumeration
       +---- serial-number                           string
       +---- idevid-issuer?                          binary
       +---- pinned-domain-cert                      binary
       +---- domain-cert-revocation-checks?          boolean
       +---- nonce?                                  binary
       +---- last-renewal-date?
       |       yang:date-and-time
       +---- pinned-domain-subject-public-key-info?   binary
```

### 6.4. SID values

[EDNote: the appropriate generation of the SID values is under discussion]

```
      SID Assigned to
--------- --------------------------------------------------
  1001100 module ietf-cwt-voucher
  1001101 module ietf-restconf
  1001102 module ietf-voucher
  1001103 module ietf-yang-types
  1001104 data .../ietf-cwt-voucher:voucher
  1001105 data .../assertion
  1001106 data .../created-on
  1001107 data .../domain-cert-revocation-checks
  1001108 data .../expires-on
  1001109 data .../idevid-issuer
  1001110 data .../last-renewal-date
  1001111 data .../nonce
  1001112 data .../pinned-domain-cert
  1001113 data .../pinned-domain-subject-public-key-info
  1001114 data .../serial-number
```

## 6.5. YANG Module

[EDNote: the appropriate syntax of the module is under discussion]

```
<CODE BEGINS> file "ietf-cwt-voucher@2017-12-11.yang"
/* -*- c -*- */
module ietf-cwt-voucher {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-cwt-voucher";
  prefix "vcwt";

  import ietf-voucher {
    prefix "v";
  }

  organization
   "IETF 6tisch Working Group";

  contact
   "WG Web:   <http://tools.ietf.org/wg/6tisch/>
    WG List:  <mailto:6tisch@ietf.org>
    Author:   Michael Richardson
              <mailto:mcr+ietf@sandelman.ca>";

  description
   "This module defines the format for a voucher, which is produced by
    a pledge’s manufacturer or delegate (MASA) to securely assign one
    or more pledges to an ’owner’, so that the pledges may establish a
```

```
     secure connection to the owner's network infrastructure.

     This version provides a very restricted subset appropriate
     for very constrained devices.
     In particular, it assumes that nonce-ful operation is
     always required, that expiration dates are rather weak, as no
     clocks can be assumed, and that the Registrar is identified
     by a pinned Raw Public Key.

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT',
     'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in
     the module text are to be interpreted as described in RFC 2119.";

   revision "2017-12-11" {
     description
      "Initial version";
     reference
      "RFC XXXX: Voucher Profile for Constrained Devices";
   }

   // Grouping defined for future usage
   grouping voucher-cwt-grouping {
     description
       "Grouping to allow reuse/extensions in future work.";

     uses v:voucher-artifact-grouping {
       augment "voucher" {
         description "Base the CWT voucher upon the regular one";
         leaf pinned-domain-subject-public-key-info {
           type binary;
           description
             "The pinned-domain-subject replaces the
          pinned-domain-certificate in constrained uses of
          the voucher.  The pinned-domain-public-key-info is the
          Raw Public Key of the Registrar. This field is encoded
          as specified in RFC7250, section 3.
          The ECDSA algorithm MUST be supported.
          The EdDSA algorithm as specified in
          draft-ietf-tls-rfc4492bis-17 SHOULD be supported.
          Support for the DSA algorithm is not recommended.
          Support for the RSA algorithm is a MAY.";
         }
       }
     }
   }
 }
 <CODE ENDS>
```

### 6.5.1. Example voucher artifacts

TBD

## 6.6. CMS format voucher and voucher-request artifacts

The IETF evolution of PKCS#7 is CMS [RFC5652]. The CMS signed voucher is much like the equivalent voucher defined in [I-D.ietf-anima-voucher].

A different eContentType of TBD1 is used to indicate that the contents are in a different format than in [I-D.ietf-anima-voucher].

The ContentInfo structure contains a payload consisting of the CBOR encoded voucher. The [I-D.ietf-core-yang-cbor] use of delta encoding creates a canonical ordering for the keys on the wire. This canonical ordering is not important as there is no expectation that the content will be reproduced during the validation process.

Normally the recipient is the pledge and the signer is the MASA.

[I-D.ietf-anima-bootstrapping-keyinfra] supports both signed and unsigned voucher requests from the pledge to the JRC. In this specification, voucher-request artifact is not signed from the pledge to the registrar. From the JRC to the MASA, the voucher-request artifact MUST be signed by the domain owner key which is requesting ownership.

The considerations of [RFC5652] section 5.1, concerning validating CMS objects which are really PKCS7 objects (cmsVersion=1) applies.

The CMS structure SHOULD also contain all the certificates leading up to and including the signer’s trust anchor certificate known to the recipient. The inclusion of the trust anchor is unusual in many applications, but without it third parties can not accurately audit the transaction.

The CMS structure MAY also contain revocation objects for any intermediate certificate authorities (CAs) between the voucher-issuer and the trust anchor known to the recipient. However, the use of CRLs and other validity mechanisms is discouraged, as the pledge is unlikely to be able to perform online checks, and is unlikely to have a trusted clock source. As described below, the use of short-lived vouchers and/or pledge provided nonce provides a freshness guarantee.

### 6.7. COSE format voucher and voucher-request artifacts

This section to be added.

## 7. Design Considerations

The design considerations for the CBOR encoding of vouchers is much the same as for [I-D.ietf-anima-voucher].

One key difference is that the names of the leaves in the YANG does not have a material effect on the size of the resulting CBOR, as the SID translation process assigns integers to the names.

## 8. Security Considerations

### 8.1. Clock Sensitivity

TBD.

### 8.2. Protect Voucher PKI in HSM

TBD.

### 8.3. Test Domain Certificate Validity when Signing

TBD.

## 9. IANA Considerations

### 9.1. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

```
URI: urn:ietf:params:xml:ns:yang:ietf-cwt-voucher
Registrant Contact: The ANIMA WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-cwt-voucher-request
Registrant Contact: The ANIMA WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

### 9.2. The YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format defined in [RFC6020], the the following registration is requested:

```
name:         ietf-cwt-voucher
namespace:    urn:ietf:params:xml:ns:yang:ietf-cwt-voucher
prefix:       vch
reference:    RFC XXXX

name:         ietf-cwt-voucher-request
namespace:    urn:ietf:params:xml:ns:yang:ietf-cwt-voucher-request
prefix:       vch
reference:    RFC XXXX
```

### 9.3. The SMI Security for S/MIME CMS Content Type Registry

This document registers an OID in the "SMI Security for S/MIME CMS Content Type" registry (1.2.840.113549.1.9.16.1), with the value:

| Decimal | Description | References |
| --- | --- | --- |
| TBD1 | id-ct-animaCBORVoucher | [ThisRFC] |

EDNOTE: should a separate value be used for Voucher Requests?

### 9.4. The SID registry

The SID range 1001100 was allocated by comi.space to the IETF-CWT-VOUCHER yang module.

The SID range 1001150 was allocated by comi.space to the IETF-CWT-VOUCHER-REQUEST yang module.

EDNOTE: it is unclear if there is further IANA work required.

### 9.5. Media-Type Registry

This section registers the 'application/voucher-cms+cbor' media type in the "Media Types" registry. These media types are used to indicate that the content is a CBOR voucher signed with a cms structure.

Type name: application
Subtype name: voucher-cms+cbor
Required parameters: none
Optional parameters: none
Encoding considerations: CMS-signed CBOR vouchers are CBOR encoded.
Security considerations: See Security Considerations, Section
Interoperability considerations: The format is designed to be broadly interoperable.
Published specification: THIS RFC.
Applications that use this media type: ANIMA, 6tisch, and other zero-touch imprinting systems
Additional information:
Magic number(s): None
File extension(s): .cbor
Macintosh file type code(s): none
Person & email address to contact for further information: IETF ANIMA WG
Intended usage: LIMITED
Restrictions on usage: NONE
Author: ANIMA WG
Change controller: IETF
Provisional registration? (standards tree only): NO

### 9.6. CoAP Content-Format Registry

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are needed for the below media types. These can be registered either in the Expert Review range (0-255) or IETF Review range (256-9999). Addition: Type name: application Subtype name: voucher-cms+cbor ID: TBD2 Required parameters: None Optional parameters: None Encoding considerations: CBOR Security considerations: As defined in this specification Published specification: this document Applications that use this media type: ANIMA bootstrap (BRSKI)

## 10. Acknowledgements

TBD

## 11. References

### 11.1. Normative References

[I-D.ietf-ace-cbor-web-token]
Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", draft-ietf-ace-cbor-web-token-12 (work in progress), February 2018.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-09 (work in progress), October 2017.

[I-D.ietf-anima-voucher]
Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "Voucher Profile for Bootstrapping Protocols", draft-ietf-anima-voucher-07 (work in progress), January 2018.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-08 (work in progress), January 2018.

[I-D.ietf-core-sid]
Veillette, M. and A. Pelov, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-03 (work in progress), December 2017.

[I-D.ietf-core-yang-cbor]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-06 (work in progress), February 2018.

[I-D.vanderstok-ace-coap-est]
Stok, P., Kampanakis, P., Kumar, S., Richardson, M., Furuhed, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-vanderstok-ace-coap-est-04 (work in progress), January 2018.

[ieee802-1AR]
IEEE Standard, ., "IEEE 802.1AR Secure Device Identifier", 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

[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>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

## 11.2. Informative References

[duckling] Stajano, F. and R. Anderson, "The resurrecting duckling: security issues for ad-hoc wireless networks", 1999, <https://www.cl.cam.ac.uk/~fms27/papers/1999-StajanoAnd-duckling.pdf>.

[I-D.ietf-netmod-yang-tree-diagrams] Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-05 (work in progress), January 2018.

[pledge] Dictionary.com, ., "Dictionary.com Unabridged", 2015, <http://dictionary.reference.com/browse/pledge>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

## Appendix A. EST messages to EST-coaps

This section extends the examples from Appendix A of [I-D.vanderstok-ace-coap-est]. The CoAP headers are only worked out for the enrollstatus example.

## A.1. enrollstatus

A coaps enrollstatus message can be :

```
GET coaps://[192.0.2.1:8085]/est/es
```

The corresponding coap header fields are shown below.

```
Ver = 1
T = 0 (CON)
Code = 0x01 (0.01 is GET)
Options
 Option1 (Uri-Host)
   Option Delta = 0x3  (option nr = 3)
   Option Length = 0x9
   Option Value = 192.0.2.1
 Option2 (Uri-Port)
   Option Delta = 0x4  (option nr = 4+3=7)
   Option Length = 0x4
   Option Value = 8085
 Option3 (Uri-Path)
   Option Delta = 0x4   (option nr = 7+4= 11)
   Option Length = 0x7
   Option Value = /est/es
Payload = [Empty]
```

A 2.05 Content response with an unsigned JSON voucher (ct=50) will then be:

```
2.05 Content (Content-Format: application/json)
  {payload}
```

With CoAP fields and payload:

```
Ver=1
T=2 (ACK)
Code = 0x45 (2.05 Content)
Options
  Option1 (Content-Format)
  Option Delta = 0xC  (option nr 12)
  Option Length = 0x2
  Option Value = 0x32 (application/json)

  Payload =
 [EDNOTE: put here voucher payload ]
```

## A.2. voucher_status

A coaps voucher_status message can be :

```
GET coaps://[2001:db8::2:1]:61616]/est/vs
```

A 2.05 Content response with a non signed JSON voucher (ct=50) will then be:

```
2.05 Content (Content-Format: application/json)
Payload =
[EDNOTE: put here voucher payload ]
```

## A.3. requestvoucher

A coaps requestvoucher message can be :

```
GET coaps://[2001:db8::2:1]:61616]/est/rv
```

A 2.05 Content response returning CBOR voucher signed with a cms structure(ct=TBD2) will then be:

```
2.05 Content (Content-Format: application/voucher-cms+cbor)
Payload =
[EDNOTE: put here encrypted voucher payload ]
```

## A.4. requestauditing

A coaps requestauditing message can be :

```
GET coaps://[2001:db8::2:1]:61616]/est/ra
```

A 2.05 Content response with a COSE voucher (ct=16) will then be:

```
2.05 Content (Content-Format: application/cose)
Payload =
[EDNOTE: put here COSE voucher payload ]
```

## Authors' Addresses

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Peter van der Stok
vanderstok consultancy

Email: consultancy@vanderstok.org

Panos Kamapanakis
Cisco Systems

Email: pkampana@cisco.com