Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 28, 2018

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

October 25, 2017

Guidelines for Autonomic Service Agents
draft-carpenter-anima-asa-guidelines-03

Abstract

This document proposes guidelines for the design of Autonomic Service Agents for autonomic networks. It is based on the Autonomic Network Infrastructure outlined in the ANIMA reference model, making use of the Autonomic Control Plane and the Generic Autonomic Signaling Protocol.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 28, 2018.

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].

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."

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 be multi-threaded so that it can 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."

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 ASAs.

2. Logical Structure of an Autonomic Service Agent

As mentioned above, all but the simplest ASAs will be multi-threaded programs.

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 Intent.
- 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. Examples of such threads include:

- A background thread to repeatedly flood an objective to the AN, so that any ASA can receive the objective's latest value.
- A thread to accept incoming synchronization requests for an objective managed by this ASA.
- A thread to accept incoming negotiation requests for an objective managed by this ASA, and then to conduct the resulting negotiation with the counterpart ASA.
- A thread to manage subsidiary non-autonomic devices directly.

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

Note: If the programming environment does not support multi-threading, an 'event loop' style of implementation could be adopted, in which case each of the above threads would be implemented as an event handler called in turn by the main loop. In this case, the GRASP API (Section 3.3) must provide non-blocking calls. If necessary, the GRASP session identifier will be used to distinguish simultaneous negotiations.

According to the degree of parallelism needed by the application, some of these threads 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 Intent 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.

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 clients of the ACP. It 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.liu-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 4), 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 offers 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.liu-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 as the value of a GRASP objective. 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 Intent mechanism

At the time of writing, the Intent mechanism for the ANI is undefined. It is expected to operate by an information distribution mechanism 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 Intent, so every ASA implementation must include carefully chosen default values and settings for all parameters and choices that might depend on Intent.

4. 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.

5. 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.

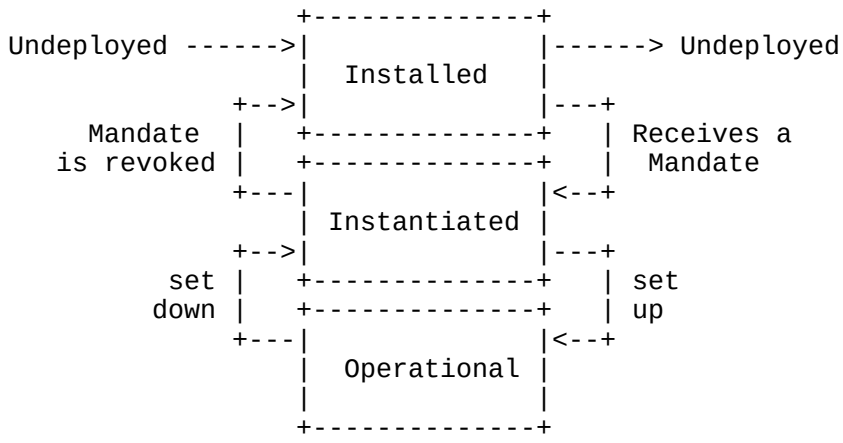


Figure 1: Life cycle of an Autonomic Service Agent

5.1. 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.

5.1.1. 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).

5.2. 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).

5.2.1. 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]

5.2.2. 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,

5.2.3. 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] - Autonomic Functions 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)

5.3. 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].

6. Coordination

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].

7. 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. 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.

6. 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.

7. 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.

8. 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).

8. 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]. 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 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.

9. IANA Considerations

This document makes no request of the IANA.

10. Acknowledgements

TBD.

11. References

11.1. Normative References

[I-D.ietf-anima-autonomic-control-plane]
Behringer, M., Eckert, T., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-ietf-anima-autonomic-control-plane-12 (work in progress), October 2017.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-08 (work in progress), October 2017.

[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.

[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>.

11.2. 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] ETSI GS AFI 002, "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. and J. 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.ciavaglia-anima-coordination] Ciavaglia, L. and P. Peloso, "Autonomic Functions Coordination", draft-ciavaglia-anima-coordination-01 (work in progress), March 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-05 (work in progress), October 2017.

[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-05 (work in progress), October 2017.

[I-D.peloso-anima-autonomic-function] Pierre, P. and L. Ciavaglia, "A Day in the Life of an Autonomic Function", draft-peloso-anima-autonomic-function-01 (work in progress), March 2016.

[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.

[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>.

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

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

Authors’ Addresses

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

Email: brian.e.carpenter@gmail.com

Laurent Ciavaglia
Nokia
Villarceaux
Nozay 91460
FR

Email: laurent.ciavaglia@nokia.com

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

Email: jiangsheng@huawei.com

Pierre Peloso
Nokia
Villarceaux
Nozay 91460
FR

Email: pierre.peloso@nokia.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 16, 2018

B. Carpenter
Univ. of Auckland
S. Jiang
B. Liu
Huawei Technologies Co., Ltd
September 12, 2017

Transferring Bulk Data over the GeneRic Autonomic Signaling Protocol (GRASP)
draft-carpenter-anima-grasp-bulk-00

Abstract

This document describes how bulk data may be transferred between Autonomic Service Agents via the GeneRic Autonomic Signaling Protocol (GRASP).

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. However, those mechanisms are limited to distributing GRASP Objective Options contained in messages that cannot exceed the GRASP maximum message size of 2048 bytes.

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.

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 rather than an additional mechanism, but there is a need to transfer a large amount of data. The basic model is to use the GRASP

Negotiation process to transfer and acknowledge multiple blocks of data in successive negotiation steps.

NOTE: This is an early draft of a solution. As the specification becomes more mature, the authors expect it to become precise enough to be placed on the standards track.

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. 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.

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 currently specify retransmission for failed blocks, so a failed transfer will need to be restarted. In an enterprise network with low bit error rates, this is not considered a serious issue.

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.

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 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 [I-D.ietf-cbor-cddl] 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.liu-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. Datagram Transport Layer

The above description and example assume that GRASP is implemented over a reliable transport layer such as TCP, such that lost or corrupted messages need not be considered. In the event that GRASP is implemented over an unreliable transport layer such as UDP, it would be necessary 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]. In such a case, a bulk transfer solution has several choices:

1. Accept the overhead of an adaptation layer, and therefore assume a network-layer MTU of 1280 bytes.

2. Attempt to determine the actual MTU available without lower-layer fragmentation.

3. Attempt to determine a message size that provides optimum performance.

TBD: further discussion?

6. Other Considerations

TBD - discussion of specific use cases?

TBD - discussion of user space API for bulk transfer?

7. 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.

8. IANA Considerations

This document makes no request of the IANA.

9. Acknowledgements

TBD.

10. References

10.1. 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.

[I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-00 (work in progress), July 2017.

10.2. Informative References

[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-04 (work in progress), July 2017.

[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-04 (work in progress), June 2017.

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

[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>.

[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-00, 2017-09-12:

Initial version.

Authors' Addresses

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

Email: brian.e.carpenter@gmail.com

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

Email: jiangsheng@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: April 15, 2018

M. Behringer, Ed.

T. Eckert, Ed.
Huawei

S. Bjarnason
Arbor Networks

October 12, 2017

An Autonomic Control Plane (ACP)
draft-ietf-anima-autonomic-control-plane-12

Abstract

Autonomic functions need a control plane to communicate, which depends on some addressing and routing. This Autonomic Management and 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 OAM (Operations Administration and Management) communications over a network that is secure and reliable even when the network is not configured, or not misconfigured.

Table of Contents

1. Introduction

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 currently being defined in the document [I-D.ietf-anima-reference-model]

Autonomic functions need an autonomously built communications infrastructure or network plane (there is no well-established name for this). 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 Management and Control". For naming consistency with that prior document, this document continues to use the name ACP though.

Today, the management and control plane of networks typically runs in the global routing table, which is dependent on correct configuration and routing. Misconfigurations or routing problems can therefore disrupt management and control channels. Traditionally, an out of band network has been used to recover from such problems, or personnel is sent on site to access devices through console ports (craft ports). 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 "Autonomic Control Plane" (ACP) which is a virtual in-band network designed to be as independent as possible of configuration, addressing and routing problems. The details how this achieved are defined in Section 6. The ACP is designed to remains operational even in the presence of configuration errors, addressing or routing issues, or where policy could inadvertently affect connectivity of both data packets or control packets.

This document uses the term "data-plane" to refer to anything in the network nodes that is not the ACP, and therefore considered to be dependent on (mis-)configuration. This data-plen includes both the traditional forwarding-plane, as well as any pre-existing control-plane, such as routing protocols that establish routing tables for the forwarding plane.

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

- Autonomic functions communicate over the ACP. The ACP therefore supports directly Autonomic Networking functions, as described in [I-D.ietf-anima-reference-model]. For example, GRASP [I-D.ietf-anima-grasp] runs securely inside the ACP and depends on the ACP as its "security and transport substrate".

- An operator can use it to log into remote devices, even if the network is misconfigured or not configured.

- A controller or network management system can use it to securely bootstrap network devices in remote locations, even if the 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]

This document describes these use cases for the ACP in Section 3, it defines the requirements in Section 4. Section 5 gives an overview how the ACP is constructed, and in Section 6 the process is defined in detail. Section 7 defines how to support ACP on L2 switches. Section 8 explains how non-ACP nodes and networks can be integrated. The following sections are non-normative: Section 7 reviews benefits of the ACP (after all the details have been defined), Section 10 provides additional explanations and describes additional details or

future work possibilities that where considered not to be appropriate for standardization in this document but nevertheless assumed to be helpful for candidate adopters of the ACP.

The ACP as defined in this document can be implemented and operated without dependency against other components of autonomous networks except for the GRASP protocol on which it depends. The document "Autonomic Network Stable Connectivity" [I-D.ietf-anima-stable-connectivity] describes how the ACP alone can be used to provide stable connectivity for autonomic and non-autonomic OAM applications ("Operations Administration and Management"). It also explains on how existing management solutions can leverage the ACP in parallel with traditional management models, when to use the ACP and, how to integrate IPv4 based management, etc.

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

2. Acronyms and Terminology

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 defined in this document. It provides secure zero-touch transitive (network wide) IPv6 connectivity for all nodes in the same ACP domain. 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 domain information field of the ACP domain 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: 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 domain certificates that allow them to authenticate each other as members of the ACP domain. See Section 6.1.2.

domain information (field): An rfc822Name information element (e.g.: field) in the domain certificate in which the ACP relevant information is encoded: the domain name and the ACP address.

ACP loopback interface: The interface in the ACP VRF that has the ACP address assigned to it.

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 prefixes routed across the ACP. In the normal/simple case, the ACP has one ULA prefix, see Section 6.10. 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 6.1.1. 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 security association established hop-by-hop between adjacent ACP nodes to carry traffic of the ACP VRF separated from data-plane traffic in-band over the same links as the data-plane.

ACP secure channel protocol: The protocol used to build an ACP secure channel, e.g.: IKEv2/IPsec or dTLS.

ACP virtual interface: An interface in the ACP VRF mapped to one or more ACP secure channels. See Section 6.12.5.

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 domain information field of the Domain Certificate. See Section 6.1.1.

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 bootstrap and stable connectivity for SDN networks - see [I-D.ietf-anima-stable-connectivity].

ANIMA: "Autonomic Networking Integrated Model and Approach". ACP, BRSKI and GRASP are products 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.

data-plane: The counterpoint to the ACP VRF in an ACP node: all VRFs other than the ACP VRF. In a simple ACP or ANI node, the data-plane is typically provisioned non-autonomic, for example manually (including across the ACP) or via SDN controllers. In a full Autonomic Network node, the data-plane is managed autonomically via Autonomic Functions and Intent. Note that other (non-ANIMA) RFC use the data-plane to refer to what is better called the forwarding plane. This is not the way the term is used in this document!

ACP (ANI/AN) Domain Certificate: A provisioned certificate (LDevID) carrying the domain information field 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.

device: A physical system, or physical node.

Enrollment: The process where a node presents identification (for example through keying material such as the private key of an IDevID) to a network and acquires a network specific identity and trust anchor such as an LDevID.

EST: "Enrollment over Secure Transport" ([RFC7030]). IETF standard protocol for enrollment of a node with an LDevID. 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" which is run inside the ACP to support BRSKI and other future 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].

Intent: Northbound operator and automation facing interface of an Autonomic Network according to [I-D.ietf-anima-reference-model].

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

MIC: "Manufacturer Installed Certificate". Another word not used in this document to describe an IDevID.

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.

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

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

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

sUDI: "secured Unique Device Identifier". Another term not used in this document to refer to an IDevID.

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: A "Unique Local Address" (ULA) is an IPv6 address in the block fc00::/7, defined in [RFC4193]. It is the approximate IPv6 counterpart of the IPv4 private address ([RFC1918]).

(ACP) VRF: The ACP is modelled in this document as a "Virtual Routing and Forwarding" (VRF) component in a network node.

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 [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

3. Use Cases for an Autonomic Control Plane

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. This way, there is only need for a single discovery mechanism, a single security mechanism, and 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 can happen without requiring any configuration such as the transit connectivity to

bootstrap further devices. A new device can automatically be bootstrapped in a secure fashion and be deployed with a domain certificate. This does not require any configuration on intermediate nodes, because they can communicate zero-touch and securely through the ACP.

3.3. Data-Plane Independent Permanent Reachability

Today, most critical control plane protocols and network management protocols are running in the data-plane (global routing table) of the network. This leads to undesirable dependencies between control and management plane on one side and the data-plane on the other: Only if the data-plane is operational, will the other planes 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 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 console access can help recover from such issues.

Data-plane dependencies also affect applications in a NOC ("Network Operations Center") such as SDN controller applications: Certain network changes are today hard to operate, 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.

The ACP provides reachability that is independent of the data-plane (except for the dependency discussed in Section 6.12.2 which can be removed through future work), which allows control plane and management 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 configuration or global routing table.

- 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 "Autonomic Network Stable Connectivity" [I-D.ietf-anima-stable-connectivity] explains the use cases for the ACP in significantly more detail and explains how the ACP can be used in practical network operations.

4. Requirements

The Autonomic Control Plane has the following 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, security (can block easily at edge).

ACP3: The ACP MUST use autonomically managed address space. Reason: easy bootstrap and setup ("autonomic"); robustness (admin can’t mess things up so easily). This document suggests to use ULA addressing for this purpose ("Unique Local Address", see [RFC4193]).

ACP4: The ACP MUST be generic. Usable by all the functions and protocols of the AN infrastructure. It 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 SHOULD (very strong SHOULD) be encrypted.

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

The Autonomic Control Plane is constructed in the following way (for details, see Section 6):

1. An ACP node creates a VRF ("Virtual Routing and Forwarding") instance, or a similar virtual context.

2. It determines, following a policy, a candidate peer list. This is the list of nodes to which it should establish an Autonomic Control Plane. Default policy is: To all link-layer adjacent nodes supporting ACP.

3. For each node in the candidate peer list, it authenticates that node and negotiates a mutually acceptable channel type.

4. It then establishes a secure tunnel of the negotiated channel type. These tunnels are placed into the previously set up VRF. This creates an overlay network with hop-by-hop tunnels.

5. Inside the ACP VRF, each node sets up a loopback interface with its ULA IPv6 address.

6. Each node runs a lightweight routing protocol, to announce reachability of the virtual addresses inside the ACP (see Section 6.12.5).

Note:

- Non-autonomic NMS ("Network Management Systems") or SDN controllers have to be manually connected into the ACP.

- Connecting over non-ACP Layer-3 clouds initially requires a tunnel between ACP nodes.

- None of the above operations (except manual ones) is reflected in the configuration of the node.

The following figure illustrates the ACP.

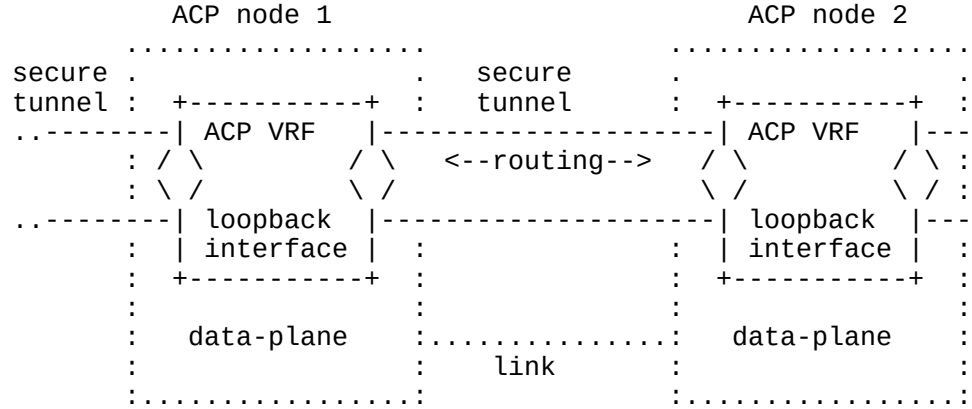


Figure 1

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. 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 describes the components and steps to set up an Autonomic Control Plane (ACP), and highlights the key properties which make it "indestructible" against many inadvertent changes to the data-plane, for example caused by misconfigurations.

An ACP node can be a router, switch, controller, NMS host, or any other IP capable node. Initially, it must have a certificate, as well as an (empty) ACP Adjacency Table (described in Section 6.2). It then can start to discover ACP neighbors and build the ACP. This is described step by step in the following sections:

6.1. ACP Domain, Certificate and Network

ACP relies on group security. An ACP domain is a group of nodes that trust each other to participate in ACP operations. To establish trust, the ACP requires certificates: An ACP node MUST have keying material consisting of a certificate (LDevID), with which it can cryptographically assert its membership in the ACP domain and trust anchor(s) associated with that certificate with which it can verify the membership of other nodes (see Section 6.1.2). The certificate is called the ACP domain certificate, the trust anchor(s) are the CA ("Certificate Authority") 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 following ACP specific information field in its domain certificate as well as those of candidate ACP peers. See Section 10.1 for more information about enrollment or provisioning options.

Note: LDevID ("Local Device IDentification") is the term used to indicate a certificate that was provisioned by the owner of a node as opposed to IDevID ("Initial Device IDentifier") that may has been loaded on the node during manufacturing time. Those IDevID do not include owner and deployment specific information to allows autonomic establishment of trust for the operations of an ACP domain (e.g.: between two ACP nodes without relying on any third party).

This document uses the term ACP in many places where its reference document use the word autonomic. This is done because those

reference document consider fully autonomic network and nodes, but support of ACP does not require support for other components of autonomic networks. Therefore the word autonomic would be irritating 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 domain 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).

The ACP domain certificate can and should be used for any authentication between ACP nodes where the required security is domain membership. Section 6.1.2 defines this "ACP domain membership check". The uses of this check that are standardized in this document are for the establishment of ACP secure channels (Section 6.6) and for ACP GRASP (Section 6.8.2). Other uses are subject to future work, but it is recommended that it is the default security check for any end-to-end connections between ASA. It is equally useable by other functions such as legacy OAM functions.

6.1.1. Certificate Domain Information Field

Information about the domain MUST be encoded in the domain certificate in a subjectAltName / rfc822Name field according to the following ABNF definition ([RFC5234]):

[RFC Editor: Please substitute SELF in all occurences of rfcSELF with the RFC number assigned to this document and remove this comment line]

```
domain-information = local-part "@" domain

local-part = key "." local-info

key = "rfcSELF"

local-info = [ acp-address ] [ "+" rsub extensions ]

acp-address = 32hex-dig

hex-dig = DIGIT / "a" / "b" / "c" / "d" / "e" / "f"
```

rsub = [domain-name] ; empty if not used

domain = domain-name

routing-subdomain = [rsub " ."] domain

domain-name = ; <domain> according to section 3.5 of [RFC1034]

extensions = *("+" extension)

extension = ; future definition. Must fit into [RFC5322] simple dot-atom format.

Example:

domain-information = rfcSELF+fda379a6f6ee00000200000064000000+area51.research@acp.example.com

routing-subdomain = area51.research.acp.example.com

"acp-address" MUST be the ACP address of the node. It is optional to support variations of the ACP mechanisms, for example other means for nodes to assign ACP addresses to themselves. Such methods are subject to future work though.

Note: "acp-address" cannot use standard IPv6 address formats because it must match the simple dot-atom format of [RFC5322]. ":" are not allowed in that format.

"domain" is used to indicate the ACP Domain across which all ACP nodes trust each other and are willing to build ACP channel to each other. See Section 6.1.2. Domain SHOULD be the FQDN of a domain owned by the operator assigning the certificate. This is a simple method to ensure that the domain is globally unique and collision of ACP addresses would therefore only happen due to ULA hash collisions. If the operator does not own any FQDN, it should choose a string in FQDN format that intends to be equally unique.

"routing-subdomain" is the autonomic subdomain that is used to calculate the hash for the ULA prefix of the ACP address of the node. "rsub" is optional and should only be used when its impacts are understood. When "rsub" is not used, "routing-subdomain" is the same as "domain".

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

Note that the maximum size of "domain-information" is 254 characters and the maximum size of node-info is 64 characters according to [RFC5280] that is referring to [RFC2821] (superseded by [RFC5321]).

The subjectAltName / rfc822Name encoding of the ACP domain name and ACP address is used for the following reasons:

- There are a wide range of pre-existing protocols/services where authentication with LDevID is desirable. Enrolling and maintaining separate LDevIDs for each of these protocols/services is often undesirable overhead. Therefore, the information element required for the ACP in the domain certificate should be encoded in a way that minimizes the possibility of creating incompatibilities with such other uses beside the authentication for the ACP.

- The elements in the LDevID required for the ACP should not cause incompatibilities with any pre-existing ASN.1 software potentially in use in those other pre-existing SW systems. This eliminates the use of novel information elements because those require extensions to those pre-existing ASN.1 parsers.

- subjectAltName / rfc822Name is a pre-existing element that must be supported by all existing ASN.1 parsers for LDevID.

- The elements in the LDevID required for the ACP should also not be misinterpreted by any pre-existing protocol/service that might use the LDevID. If the elements used for the ACP are interpreted by other protocols/services, then the impact should be benign.

- Using an IP address format encoding could result in non-benign misinterpretation of the domain information field; other protocol/ services unaware of the ACP could try to do something with the ACP address that would fail to work correctly. For example, the address could be interpreted to be an address of the node in a VRF other than the ACP VRF.

- At minimum, both the AN domain name and the non-domain name derived part of the ACP address need to be encoded in one or more appropriate fields of the certificate, so there are not many alternatives with pre-existing fields where the only possible conflicts would likely be beneficial.

- rfc822Name encoding is quite flexible. We choose to encode the full ACP address AND the domain name with sub part into a single rfc822Name information element it, so that it is easier to examine/use the "domain information field".

- The format of the rfc822Name is chosen so that an operator can set up a mailbox called rfcSELF@<domain> that would receive emails sent towards the rfc822Name of any node inside a domain. This is possible because in many modern mail systems, components behind a "+" character are considered part of a single mailbox. In other words, it is not necessary to set up a separate mailbox for every ACP node, but only one for the whole domain.

- In result, if any unexpected use of the ACP addressing information in a certificate happens, it is benign and detectable: it would be mail to that mailbox.

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

6.1.2. ACP domain membership check

The following points constitute the ACP domain membership check:

- The peer certificate is valid as proven by the security associations protocol exchange.

- The peers certificate is signed by one of the trust anchors associated with the ACP domain certificate.

- If the node certificates indicate a CDP (or OCSP) then the peer's certificate must be valid according to those criteria. e.g.: OCSP check across the ACP or not listed in the CRL retrieved from the CDP.

- The peers certificate has a syntactically valid domain information field (subjectAltName / rfc822Name) and the domain name in that peers domain information field is the same as in this ACP node certificate. Note that future Intent rules may modify this. See Section 10.7.

6.1.3. Certificate Maintenance

ACP nodes MUST support certificate renewal via EST ("Enrollment over Secure Transport", see [RFC7030]) and MAY support other mechanisms. An ACP network must have at least one ACP node supporting EST server functionality across the ACP so that EST renewal is useable. The mechanism by which the domain certificate was initially provisioned SHOULD provide a mechanism to store the URL of one EST server with its ACP address into the node for later renewal. This server does not have to be the same as the one performing the initial certificate enrolment.

ACP nodes that are EST servers MUST announce their service via GRASP in the ACP through M_FLOOD messages:

Example:

```
[M_FLOOD, 12340815, h'fda379a6f6ee0000200000064000001', 210000,
    ["SRV.est", 4, 255, "EST-TLS"],
    [O_IPv6_LOCATOR,
         h'fda379a6f6ee0000200000064000001', TCP, 80]
]
```

The formal CDDL definition is:

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

 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
 objective-value = text       ; name of the (list of) of supported
                              ; protocols: "EST-TLS" for RFC7030.
```

The objective value "SRV.est" indicates that the objective is an [RFC7030] compliant EST server.

The M_FLOOD message MUST be sent periodically. The default SHOULD be 60 seconds, the value SHOULD be operator configurable. It must be so high that the aggregate amount of periodic M_FLOODs from all flooded objectives causes only negligible traffic across the ACP. The 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.

Domain certificates SHOULD by default be renewed 50% into their lifetime. When performing renewal, the node SHOULD attempt to connect to the remembered EST server. If that fails, it SHOULD attempt to connect to EST server(s) 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.

The ACP node MUST support CRLs ("Certificate Revocation Lists") via HTTPs from one or more CDPs ("CRL Distribution Points"). These CDPs MUST be indicated in the Domain Certificate when used. If the CDP URL uses an IPv6 ULA, the ACP node will try to reach it via the ACP. In that case the ACP address in the domain certificate of the CDP as learned by the ACP node during the HTTPs TLS handshake SHOULD match that ULA address in the HTTPs URL.

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 cause 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.

Certificate lifetime should be set to be as short as feasible. Given how certificate renewal is fully automated via ACP and EST, the primarily limiting factor for shorter certificate lifetimes (than the typical one year) is load on the EST server(s) and CA. It is therefore recommended that ACP domain certificates are managed via a CA chain where the assigning CA has enough performance to manage short lived certificates.

See Section 10.1 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. 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, Link-local IPv6 address (discovered by GRASP as explained below), domain, certificate. An ACP node MUST maintain this adjacency table up to date. This table is used to determine to which neighbor an ACP connection is established.

Where the next ACP node is not directly adjacent, the information in the adjacency table can be supplemented by configuration. For example, the node-ID and IP address could be configured.

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 authenticating the peer.

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.

Interaction between ACP and other autonomic elements like GRASP (see below) or ASAs should be via an API that allows (appropriately access controlled) read/write access to the ACP Adjacency Table. Specification of such an API is subject to future work.

6.3. Neighbor Discovery with DULL GRASP

The ACP uses one instance of DULL GRASP (See section 3.5.2.2 of [I-D.ietf-anima-grasp] for its formal definition) for every physical L2 subnet of the ACP node to discover physically adjacent candidate ACP neighbors. Native interfaces (e.g.: physical interfaces on physical nodes) SHOULD be brought up automatically enough so that 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 link-local address assignment (SLAAC) and DULL GRASP works 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 MLD ([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, non-ACP enabled L2 switches - because those 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 10.3 for further, non-normative suggestions 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 (see Section 10.1) then the above considerations also apply to 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:

```
      Example:

      [M_FLOOD, 12340815, h'fe80000000000000c0011001FEEF0000, 180000,
          ["AN_ACP", 4, 1, "IKEv2"],
          [O_IPv6_LOCATOR,
               h'fe80000000000000c0011001FEEF0000, UDP, 15000]
      ]
```

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 = text ; name of the (list of) secure
                               ; channel negotiation protocol(s)
```

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 (recommended) example the period of sending of the objective could be 60 seconds the indicated ttl of 180000 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 must still 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 'objective-value' parameter is (normally) a string indicating the secure channel protocol available at the specified or implied locator.

The locator 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 abbreviation for "Internet Key Exchange protocol version 2", as defined in [RFC7296]. It is the main protocol used by the Internet IP security architecture ("IPsec", see [RFC4301]). We therefore use the term "IKEv2" and not "IPsec" in the GRASP definitions below and example above. "IKEv2" has a well-defined port number 500, but in the above example, the candidate ACP neighbor is offering ACP secure channel negotiation via IKEv2 on port 15000 (for the sake of creating a non-standard example).

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 needs to flood multiple versions of the "AN_ACP" objective, each accompanied by its own locator. This can be in a single GRASP M_FLOOD message.

If multiple secure channel protocols are supported that all are run on well-defined ports, then they can be announced via a single AN_ACP objective using a list of string names as the objective value without a following locator-option.

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 6.2 which then drives the further building of the ACP to that neighbor.

6.4. Candidate ACP Neighbor Selection

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

The ACP is by default established exclusively between nodes in the same domain. This includes all routing subdomains. Section 10.7 explains how ACP connections across multiple routing subdomains are special.

Future extensions to this document including Intent can change this default behavior. Examples include:

- Build the ACP across all domains that have a common parent domain. For example ACP nodes with domain "example.com", nodes of "example.com", "access.example.com", "core.example.com" and "city.core.example.com" could all establish one single ACP.

- ACP connections across domains with different CA (certificate authorities) could establish a common ACP by installing the alternate domains' CA into the trusted anchor store. This is an executive management action that could easily be accomplished through the control channel created by the ACP.

Since Intent is transported over the ACP, the first ACP connection a node establishes is always following the default behavior. See Section 10.7 for more details.

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.

6.5. 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.

At this time in the lifecycle of ACP nodes, it is unclear whether it is feasible to even decide on a single MTI (mandatory to implement) security association protocol across all ACP nodes:

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 ("datagram Transport Layer Security version 1.2", see [RFC6347]) because that code exists already on them for end-to-end security, but low-end in-ceiling L2 switches may only want to support 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.

To support extensible secure channel protocol selection without a single common 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 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 secure channel protocol succeeds, the two peers know each other’s certificates because it must be used by all secure channel protocols for mutual authentication. The node with the lower Node-ID in the ACP address becomes Bob, the one with the higher Node-ID in the certificate Alice.

Bob becomes passive, he does not attempt to further initiate ACP secure channel protocols with Alice and does not consider it to be an error when Alice closes secure channels. Alice becomes the active party, continues to attempt setting up secure channel protocols with Bob until she arrives at the best one from her view that also works with Bob.

For example, originally Bob could have been the initiator of one ACP secure channel protocol that Bob prefers and the security association succeeded. The roles of Bob and Alice are then assigned. At this stage, the protocol may not even have completed negotiating a common security profile. The protocol could for example could have been IPsec via IKEv2 ("IP security", see [RFC4301] and "Internet Key Exchange protocol version 2", see [RFC7296]. It is now up to Alice to decide how to proceed. Even if the IPsec connecting determined a working profile with Bob, Alice might prefer some other secure protocol (e.g.: dTLS) and try to set that up with Bob. If that succeeds, she would close the IPsec connection. If no better protocol attempt succeeds, she would keep the IPsec connection.

All this negotiation is in the context of an "L2 interface". Alice and Bob 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.

6.6. 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 6.1.2. If it fails, the connection attempt is aborted and an error logged (with throttling).

6.7. Security Association protocols

The following sections define the security association protocols that we consider to be important and feasible to specify in this document:

6.7.1. ACP via IKEv2

An ACP node announces its ability to support IKEv2 as the ACP secure channel protocol in GRASP as "IKEv2".

6.7.1.1. Native IPsec

To run ACP via IPsec natively, no further IANA assignments/ definitions are required. An ACP node supporting native IPsec MUST use IPsec security setup via IKEv2, tunnel mode, local and peer link-local IPv6 addresses used for encapsulation, ESP with AES256 for encryption and SHA256 hash.

In terms of IKEv2, this means the initiator will offer to support IPsec tunnel mode with next protocol equal 41 (IPv6).

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, it would only be possible to send packets originated by the ACP node itself.

ESP is used because ACP mandates the use of encryption for ACP secure channels.

6.7.1.2. 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.

To run ACP via GRE/IPsec, no further IANA assignments/definitions are required. The ACP node MUST support IPsec security setup via IKEv2, IPsec transport mode, local and peer link-local IPv6 addresses used for encapsuation, ESP with AES256 encryption and SHA256 hash.

When GRE is used, transport mode is sufficient because the routed ACP packets are not "tunneled" by IPsec but rather by GRE: IPsec only has to deal with the GRE/IP packet which always uses the local and peer link-local IPv6 addresses and is therefore applicable to transport mode.

ESP is used because ACP mandates the use of encryption for ACP secure channels.

In terms of IKEv2 negotiation, this means the initiator must offer to support IPsec transport mode with next protocol equal to GRE (47) followed by the offer for native IPsec as described above (because that option is mandatory to support).

If IKEv2 initiator and responder support GRE, it will be selected. The version of GRE to be used must the according to [RFC7676].

6.7.2. ACP via dTLS

We define the use of ACP via dTLS in the assumption that it is likely the first transport encryption code basis supported in some classes of constrained devices.

To run ACP via UDP and dTLS v1.2 [RFC6347] a locally assigned UDP port is used that is announced as a parameter in the GRASP AN_ACP objective to candidate neighbors. All ACP nodes supporting dTLS as a secure channel protocol MUST support AES256 encryption and not permit weaker crypto options.

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. Any dTLS defined security

association mechanisms such as re-keying are used as they would be for any transport application relying solely on dTLS.

6.7.3. ACP Secure Channel Requirements

A baseline ACP node MUST support IPsec natively and MAY support IPsec via GRE. A constrained ACP node MUST support dTLS. ACP nodes connecting constrained areas with baseline areas MUST therefore support IPsec and dTLS.

ACP nodes need to specify in documentation the set of secure ACP mechanisms they support.

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. Note that this is not standard behavior in secure channel protocols such as IPsec because the certificate authentication only influences the setup of the secure channel in these protocols.

6.8. GRASP in the ACP

6.8.1. 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. They function in GRASP that makes it fundamental as a service is the ability for ACP wide service discovery (called objectives in GRASP). In most other solution designs such distributed discovery does not exist at all or was added as an afterthought and relied upon inconsistently.

ACP provides IP unicast routing via the RPL routing protocol (described below).

The ACP does not use IP multicast routing nor does it provide generic IP multicast services. 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).

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 autoconfiguring solution for ASM (Any Source Multicast - original form of IP multicast defined in [RFC1112]), which would be quite complex and

difficult to justify. One aspect of complexity that has never been attempted to be solved in IETF documents is the automatic-selection of routers that should be 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.

Future ASA may need high performance in-network data replication. That is the case when the use of IP multicast is justified. These 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 constrained flooded 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 loop-free distribution trees (DODAG) to make GRASPs flooding more efficient both for M_FLOOD and M_DISCOVERY) See Section 6.12.5 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.

6.8.2. ACP as the Security and Transport substrate for GRASP

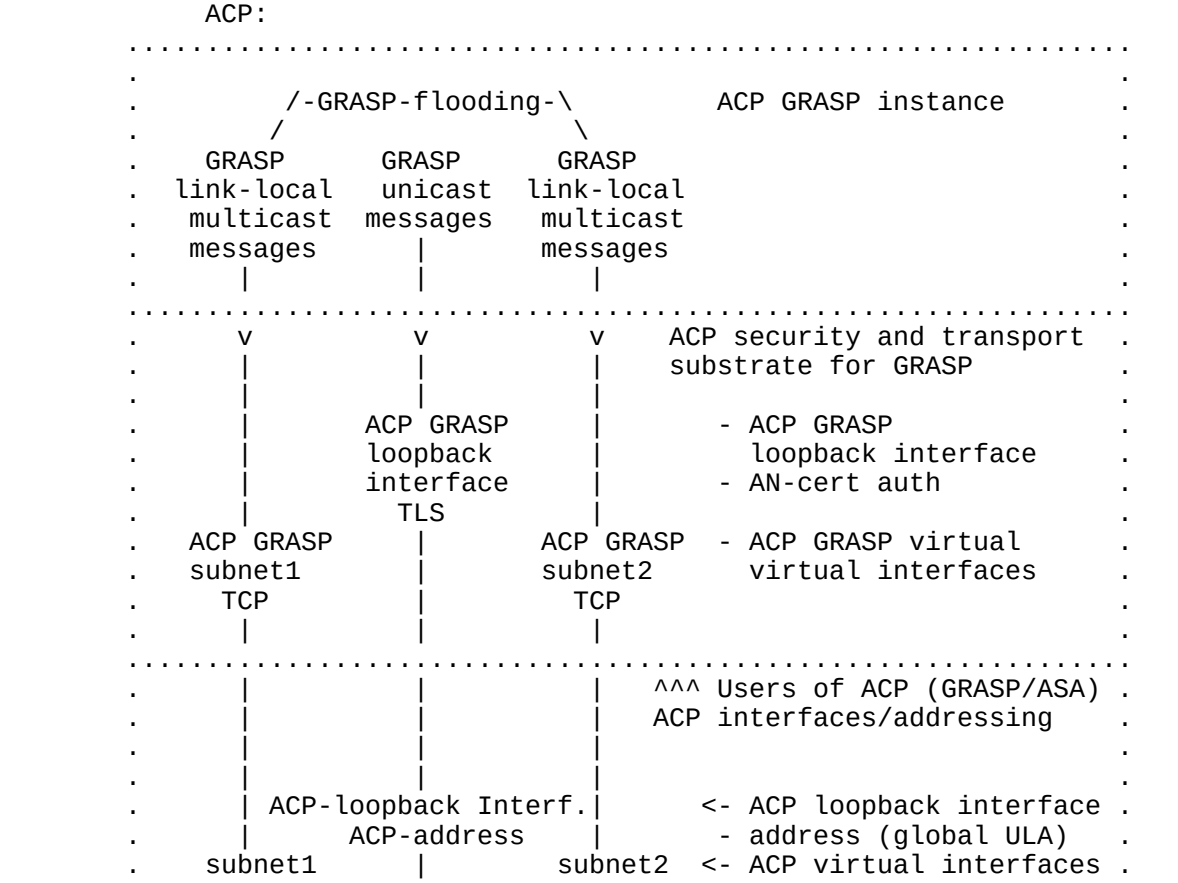
In the terminology of GRASP ([I-D.ietf-anima-grasp]), the ACP is the security and transport substrate for the GRASP instance run inside the ACP ("ACP GRASP").

This means that the ACP is responsible to ensure 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. 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 can not operate.

The way ACP acts as the security and transport substrate for GRASP is visualized in the following picture:

[RFC Editor: please try to put the following picture on a single page and remove this note. We cannot figure out how to do this with XML. The picture does fit on a single page.]



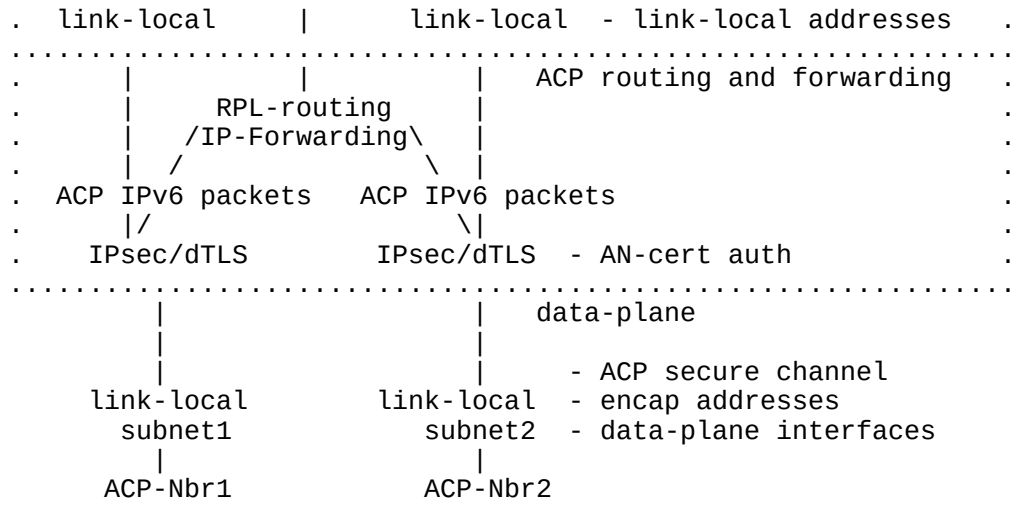


Figure 2

GRASP unicast messages inside the ACP always use the ACP address. Link-local ACP addresses must not be used inside objectives. GRASP unicast messages inside the ACP are transported via TLS 1.2 ([RFC5246]) connections with AES256 encryption and SHA256. Mutual authentication uses the ACP domain membership check defined in (Section 6.1.2).

GRASP link-local multicast messages are targeted for a specific ACP virtual interface (as defined Section 6.12.5) but are sent by the ACP into an equally built ACP GRASP virtual interface 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.

TLS 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.

6.8.2.1. 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. This would result in more traffic flooding than hop-by-hop reliable retransmission as provided for by TCP.

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 would just use 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 semantic of the GRASP negotiation to an extend that allows it to proxy it without being detected, but in an AN environment this is quite likely public knowledge or evens standardized.

The GRASP TLS connections are run like any other ACP traffic through the ACP secure channels. This leads to double authentication/ encryption. Future work optimizations could avoid this but it is unclear how beneficial/feasible this is:

- The security considerations for GRASP change against attacks from non-ACP (e.g.: "outside") nodes: TLS is subject to reset attacks while secure channel protocols may be not (e.g.: IPsec is not).

- The secure channel method may leverage hardware acceleration and there may be little or no gain in eliminating it.

- The GRASP TLS connections need to implement any additional security options that are required for secure channels. For example the closing of connections when the peers certificate has expired.

6.9. 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 systems, a dedicated so called "Virtual routing and forwarding instance" (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 global routing table. Also, configuration errors from the data-plane setup do not affect the ACP.

6.10. 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 must create a loopback interface with an ACP network wide unique address inside the ACP context (as explained in in Section 6.9). 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 global ID prefix. Conversely, ULA global IDs from different domains are unlikely to clash, such that two networks can be merged, as long as the policy allows that merge. See also Section 9.1 for a discussion on merging domains.

Links inside the ACP only use link-local IPv6 addressing, such that each node only requires one routable virtual address.

6.10.1. 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 Unique Local Addresses (ULA), as specified in [RFC4193]. An alternative scheme was discussed, using assigned ULA addressing. The consensus was to use ULA-random [[RFC4193] with L=1], because it was deemed to be sufficient.

- 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 devices. 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 no not require IPv4: The ACP may carry OAM protocols. All relevant protocols (SNMP, TFTP, SSH, SCP, Radius, Diameter, ...) are available in IPv6.

6.10.2. The ACP Addressing Base Scheme

The Base ULA addressing scheme for ACP nodes has the following format:

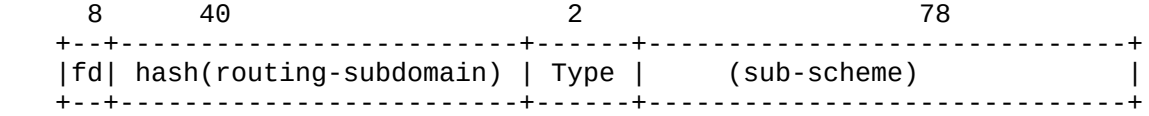


Figure 3: ACP Addressing Base Scheme

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 domain information field of domain certificates are the first 40 bits of the SHA256 hash of the routing subdomain from the same domain information field. In the example of Section 6.1.1, the routing subdomain is "area51.research.acp.example.com" and the 40 bits ULA "global ID" a379a6f6ee.
- 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 registrar will create the domain information field in response to the CSR Attribute Request by the pledge.
- 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.

6.10.3. ACP Zone Addressing Sub-Scheme

The sub-scheme defined here is defined by the Type value 00b (zero) in the base scheme.

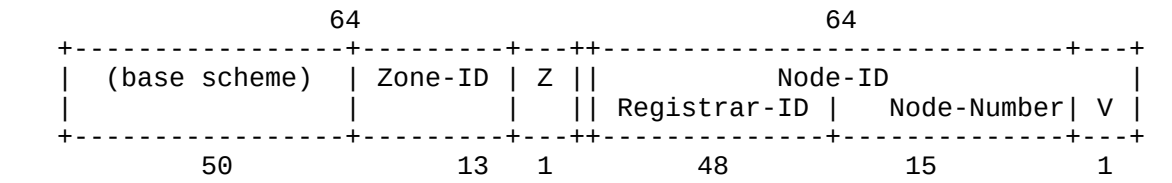


Figure 4: ACP Zone Addressing Sub-Scheme

The fields are defined as follows:

- Zone-ID: If set to all zero bits: The Node-ID bits are used as an identifier (as opposed to a locator). This results in a non-hierarchical, flat addressing scheme. Any other value indicates a zone. See Section 6.10.3.1 on how this field is used in detail.

- Z: MUST be 0.

- Node-ID: A unique value for each node.

The 64 bit Node-ID is derived and composed as follows:

- Registrar-ID (48 bit): A number unique inside the domain that identifies the registrar which assigned the Node-ID to the node. A MAC address of the registrar can be used for this purpose.

- Node-Number: A number which is unique for a given registrar, to identify the node. This can be a sequentially assigned number.

- 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 Zone addressing sub-scheme, the ACP address in the certificate has Zone and V fields as all zero bits. The ACP address set includes addresses with any Zone value and any V value.

The "Node-ID" itself is unique in a domain (i.e., the Zone-ID is not required for uniqueness). Therefore, a node can be addressed either as part of a flat hierarchy (zone ID = 0), or with an aggregation scheme (any other zone ID). A address with zone-ID = 0 is an identifier, with another zone-ID as a locator. See Section 6.10.3.1 for a description of the zone bits.

The Virtual bit in this sub-scheme allows to easily add 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 (autonomous 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 SW design of ASA (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 to announce 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.

6.10.3.1. Usage of the Zone Field

The "Zone-ID" allows for the introduction of structure in the addressing scheme.

Zone = zero is the default addressing scheme in an ACP domain. Every ACP node MUST respond to its ACP address with zone=0. Used on its own this leads to a non-hierarchical address scheme, which is suitable for networks up to a certain size. In this case, the addresses primarily act as identifiers for the nodes, and aggregation is not possible.

If aggregation is required, the 13 bit value allows for up to 8192 zones. The allocation of zone numbers may either happen automatically through a to-be-defined algorithm; or it could be configured and maintained manually.

If a node learns through an autonomic method or through configuration that it is part of a zone, it MUST also respond to its ACP address with that zone number. In this case the ACP loopback is configured with two ACP addresses: One for zone 0 and one for the assigned zone. This method allows for a smooth transition between a flat addressing scheme and an hierarchical one.

(Theoretically, the 13 bits for the Zone-ID would allow also for two levels of zones, introducing a sub-hierarchy. We do not think this is required at this point, but a new type could be used in the future to support such a scheme.)

Note: The Zone-ID is one method to introduce structure or hierarchy into the ACP. Another way is the use of the routing subdomain field in the ACP that leads to different /40 ULA prefixes within an ACP domain. This gives future work two options to consider.

6.10.4. ACP Manual Addressing Sub-Scheme

The sub-scheme defined here is defined by the Type value 00b (zero) in the base scheme.

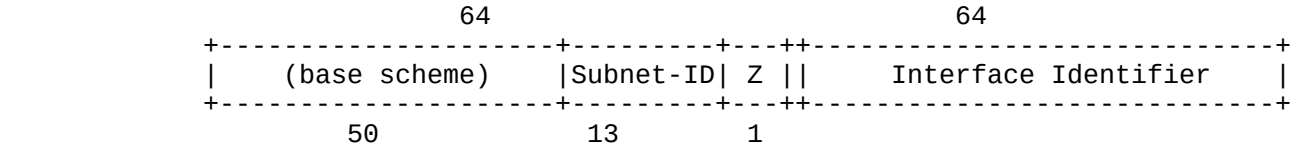


Figure 5: ACP Manual Addressing Sub-Scheme

The fields are defined as follows:

- Subnet-ID: Configured subnet identifier.
- Z: MUST be 1.
- 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). Future work may define mechanisms for auto-coordination between ACP nodes and auto-allocation of Subnet-IDs between them.

The Z field is following the Subnet-ID field so that future work could allocate/coordinate both Zone-ID and Subnet-ID consistently and use an integrated aggregatable routing approach across them. Z=0 (Zone sub-scheme) would then be used for network wide unique, registrar assigned (and certificate protected) Node-IDs primarily for ACP nodes while Z=1 would be used for node-level assigned Interface Identifiers primarily for non-ACP-nodes (on logical subnets where the ACP node is a router).

Manual addressing sub-scheme addresses SHOULD only be used in domain certificates assigned to nodes that cannot fully participate in the automatic establishment of ACP secure channels or ACP routing. The intended use are nodes connecting to the ACP via an ACP edge node and

ACP connect (see Section 8.1) - such as legacy NOC equipment. They would not use their domain certificate for ACP secure channel creation and therefore do not need to participate in ACP routing either. They would use the certificate for authentication of any transport services. The value of the Interface Identifier is left for future definitions.

6.10.5. ACP Vlong Addressing Sub-Scheme

The sub-scheme defined here is defined by the Type value 01b (one) in the base scheme.

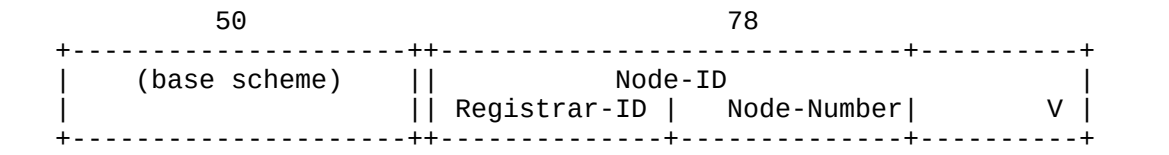


Figure 6: ACP Vlong Addressing Sub-Scheme

This addressing scheme foregoes the Zone field to allow for larger, flatter routed networks (e.g.: as in IoT) with more than 2^32 Node-Numbers. It also allows for up to 2^16 - 65536 different virtualized addresses, which could be used to address individual software components in an ACP node.

The fields are the same as in the Zone sub-scheme with the following refinements:

- V: Virtualization bit: Values 0 and 1 as in Zone sub-scheme, further values use via definition in future work.
- Registrar-ID: To maximize Node-Number and V, the Registrar-ID is reduced to 46 bits. This still allows to use the MAC address of a registrar by removing the V and U bits from the 48 bits of a MAC address (those two bits are never unique, so they cannot be used to distinguish MAC addresses).
- If the first bit of the "Node-Number" is "1", then the Node-Number is 17 bit long and the V field is 16 bit long. Otherwise the Node-Number is 33 bit long and the V field is 8 bit long. "0" bit Node-Numbers 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) nof the ACP that require separate V(irtual) addresses. "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.

6.10.6. 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-schemes is intended to enable optimized routing in large networks by reserving bits for zones. 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.

IANA is asked need to assign a new "type" for each new addressing sub-scheme. With the current allocations, only 2 more schemes are possible, so the last addressing scheme should consider to be extensible in itself (e.g.: by reserving bits from it for further extensions.

6.11. 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 the global routing table 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 manual configuration is required.

All routing updates are automatically secured in transit as the channels of the autonomic control plane are by default secured, and this routing runs only inside the ACP.

The routing protocol inside the ACP is RPL ([RFC6550]). See Section 10.5 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 Section 10.7 for more details.

6.11.1. 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 draft-ietf-roll-applicability-template.

6.11.1.1. Summary

In summary, the profile chosen for RPL is one that expects a fairly reliable network reasonable fast links so that RPL convergence will be triggered immediately upon recognition of link failure/recovery.

The key limitation of the chosen profile is that it is designed to not require any data-plane artifacts (such as [RFC6553]). While the senders/receivers of ACP packets can be legacy NOC devices connected via "ACP connect" (see Section 8.1.1 to the ACP, their connectivity can be handled as non-RPL-aware leafs (or "Internet") according to the data-plane architecture explained in [I-D.ietf-roll-useofrplinfo]. This non-artifact profile is largely driven by the desire to avoid introducing the required Hop-by-Hop headers into the ACP VRF control plane. Many devices will have their VRF forwarding code designed into silicon.

In this profile choice, RPL has no data-plane artifacts. A simple destination prefix based upon the routing table is used. A consequence of supporting only a single instanceID (containing one DODAG), the ACP will only accommodate only a single class of routing table and cannot create optimized routing paths to accomplish latency or energy goals.

Consider a network that has multiple NOCs in different locations. Only one NOC will become the DODAG root. Other NOCs will have to send traffic through the DODAG (tree) rooted in the primary NOC. Depending on topology, this can be an annoyance from a latency point of view, but it does not represent a single point of failure, as the DODAG can reconfigure itself when it detects data plane forwarding failures.

The lack of a RPI (the header defined by [RFC6553]), means that the data-plane will have no rank value that can be used to detect loops. As a result, traffic may loop until the TTL of the packet reaches zero. This the same behavior as that of other IGPs that do not have the data-plane options as RPPL. There are a variety of heuristics

that can be used to signal from the data-plane to the RPL control plane that a new route is needed.

Additionally, failed ACP tunnels will be detected by IKEv2 Dead Peer Detection (which can function as a replacement for an LLN's ETX). A failure of an ACP tunnel should signal the RPL control plane to pick a different parent.

Future Extensions to this 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. A decision for the preferred technology would have to be done when such extension is defined.

6.11.1.2. RPL Instances

Single RPL instance. Default RPLInstanceID = 0.

6.11.1.3. Storing vs. Non-Storing Mode

RPL Mode of Operations (MOP): mode 3 "Storing Mode of Operations with multicast support". Implementations should support also other modes. Note: Root indicates mode in DIO flow.

6.11.1.4. 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.

6.11.1.5. Path Metric

Hopcount.

6.11.1.6. 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)

6.11.1.7. DODAG Repair

Global Repair: we assume stable links and ranks (metrics), so no need to periodically rebuild DODAG. DODAG version only incremented under catastrophic events (e.g.: administrative action).

Local Repair: As soon as link breakage is detected, send No-Path DAO for all the targets that where reachable only via this link. As soon as link repair is detected, validate if this link provides you a better parent. If so, compute your new rank, and send new DIO that advertises your new rank. Then send a DAO with a new path sequence about yourself.

stretch_rank: none provided ("not stretched").

Data Path Validation: Not used.

Trickle: Not used.

6.11.1.8. Multicast

Not used yet but possible because of the selected mode of operations.

6.11.1.9. Security

[RFC6550] security not used, substituted by ACP security.

6.11.1.10. P2P communications

Not used.

6.11.1.11. IPv6 address configuration

Every ACP node (RPL node) announces an IPv6 prefix covering the address(es) used in the ACP node. The prefix length depends on the chosen addressing sub-scheme of the ACP address provisioned into the certificate of the ACP node, e.g.: /127 for Zone addressing sub-scheme or /112 or /120 for Vlong addressing sub-scheme. See Section 6.10 for more details.

Every ACP node MUST install a black hole (aka null) route for whatever ACP address space that it advertises (i.e.: the /96 or /127). This is avoid routing loops for addresses that an ACP node has not (yet) used.

6.11.1.12. Administrative parameters

Administrative Preference ([RFC6552], 3.2.6 - to become root): Indicated in DODAGPreference field of DIO message.

- Explicit configured "root": 0b100
- Registrar (Default): 0b011
- AN-connect (non-registrar): 0b010
- Default: 0b001.

6.11.1.13. RPL Data-Plane artifacts

RPI (RPL Packet Information [RFC6553]): Not used as there is only a single instance, and data path validation is not being used.

SRH (RPL Source Routing - RFC6552): Not used. Storing mode is being used.

6.11.1.14. 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.

6.12. 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 6.11.

6.12.1. 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, but that determination is for future work. 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. See [I-D.ietf-anima-stable-connectivity] for more details.

6.12.2. Addressing of Secure Channels in the data-plane

In order to be independent of the data-plane configuration of global IPv6 subnet addresses (that may not exist when the ACP is brought up), Link-local secure channels MUST use IPv6 link local addresses between adjacent neighbors. The fully autonomic mechanisms in this document only specify these link-local secure channels. Section 8.2 specifies extensions in which secure channels are tunnels. For those, this requirement does not apply.

The Link-local secure channels specified in this document therefore depend on basic IPv6 link-local functionality to be auto-enabled by the ACP and prohibiting the data-plane from disabling it. The ACP also depends on being able to operate the secure channel protocol (e.g.: IPsec / dTLS) across IPv6 link-local addresses, something that may be an uncommon profile. Functionally, these are the only interactions with the data-plane that the ACP needs to have.

To mitigate these interactions with the data-plane, extensions to this document may specify additional layer 2 or layer encapsulations for ACP secure channels as well as other protocols to auto-discover peer endpoints for such encapsulations (e.g.: tunneling across L3 or use of L2 only encapsulations).

6.12.3. 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 a standard issue with tunneling, not specific to running the ACP across it. Transport stacks running across ACP can perform normal PMTUD (Path MTU Discovery). Because the ACP is meant to be 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.

6.12.4. 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 6.5 allows Alice (the node with the higher ACP address) to drop all but the desired ACP channels to Bob - and Bob will not re-try to build these secure channels from his side unless Alice shows up with a previously unknown GRASP announcement (e.g.: on a different link or with a different address announced in GRASP).

6.12.5. 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.

The term "loopback interface" was introduced initially to refer to an internal interface on a node that would allow IP traffic between transport endpoints on the node in the absence or failure of any or all external interfaces, see [RFC4291] section 2.5.3.

Even though loopback interfaces where originally designed to hold only loopback addresses not reachable from outside the node, these interfaces are also commonly used today to hold addresses reachable from the outside. They are meant to be reachable independent of any external interface being operational, and therefore to be more resilient. These addresses on loopback interfaces can be thought of as "node addresses" instead of "interface addresses", and that is what ACP address(es) are. This construct makes it therefore possible to address ACP nodes with a well-defined set of addresses independent of the number of external interfaces.

For these reason, the ACP (ULA) address(es) are assigned to loopback interface(s).

ACP secure channels, e.g.: IPsec, dTLS or other future security associations with neighboring ACP nodes can be mapped to ACP virtual interfaces in different ways:

ACP point-to-point virtual interface:

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.

ACP multi-access virtual interface:

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.

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.

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.

ACP nodes MUST NOT derive their ACP virtual interface IPv6 link local address from their IPv6 link-local address used on the underlying interface (e.g.: the address that is used as the encapsulation address in the ACP secure channel protocols defined in this document). This ensures that the ACP virtual interface operations will not depend on the specifics of the encapsulation used by the ACP secure channel and that attacks against SLAAC on the physical interface will not introduce new attack vectors against the operations of the ACP virtual interface.

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 one point-to-point virtual interface to Bob and one to Carol, The sending applications itself will send two copies, if Alice's ACP emulates a LAN, GRASP will send one packet and the ACP will replicate it. 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 Charlies 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 6.8.2 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. These requirements can be avoided by coupling the ACP flooding mechanism for GRASP messages directly to RPL (flood GRASP across DODAG), but such an enhancement is subject for future work.

Care must also be taken when creating multi-access ACP virtual interfaces across ACP secure channels between ACP nodes in different domains or routing subdomains. The policies to be negotiated may be described as peer-to-peer policies in which case it is easier to create ACP point-to-point virtual interfaces for these secure channels.

7. ACP support on L2 switches/ports (Normative)

7.1. Why

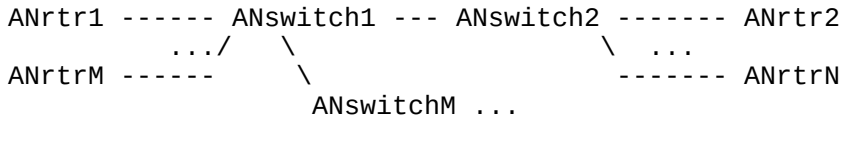


Figure 7

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 like this can creates 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 like 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.

If the device with L2 ports is supporting per L2 port ACP DULL GRASP as well as MLD snooping ([RFC4541]), then MLD snooping must be changed to never forward packets for ALL_GRASP_NEIGHBORS because that would cause the problem that per L2 port ACP DULL GRASP is meant to overcome (forwarding DULL GRASP packets across L2 ports).

The rest of ACP operations can operate in the same way as in L3 devices: Assume for example that the device is an L3/L2 hybrid device where L3 interfaces are assigned to VLANs and each VLAN has potentially multiple ports. DULL GRASP is run as described individually on each L2 port. When it discovers a candidate ACP neighbor, it passes its IPv6 link-local address and supported secure channel protocols to the ACP secure channel negotiation that can be bound to the L3 (VLAN) interface. It will simply use link-local IPv6 multicast packets to the candidate ACP neighbor. Once a secure channel is established to such a neighbor, the virtual interface to

which this secure channel is mapped should then actually be the L2 port and not the L3 interface to best map the actual physical topology into the ACP virtual interfaces. See Section 6.12.5 for more details about how to map secure channels into ACP virtual interfaces. Note that a single L2 port can still have multiple ACP neighbors if it connect for example to multiple ACP neighbors via a non-ACP enabled switch. The per L2 port ACP virtual interface can therefore still be a multi-access virtual LAN.

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.

This L2/L3 optimized approach is subject to "address stealing", e.g.: where a device on one port uses addresses of a device on another port. This is a generic issue in L2 LANs and switches 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 duplicates. This type of function needs to be enabled to prohibit DoS attacks. 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.

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, but the need to support secure channel protocols may be a limiting factor to supporting ACP on such devices. Future options such as 802.1ae could improve that situation.

A generic issue with ACP in L2 switched networks is the interaction with the Spanning Tree Protocol. Ideally, the ACP 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 for this. Instead they would simply have the ACP run across the active STP topology and the ACP would equally be interrupted and re-converge with STP changes.

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 must 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, just like 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 must support "ACP connect" (sometimes also connect "autonomic connect"):

"ACP connect" is a function on an autonomic node that is called an "ACP edge node". With "ACP connect", interfaces on the node can be configured to be put into the ACP VRF. The ACP is then accessible to other (NOC) systems on such an interface without those systems 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 encryption/novel-signaling).

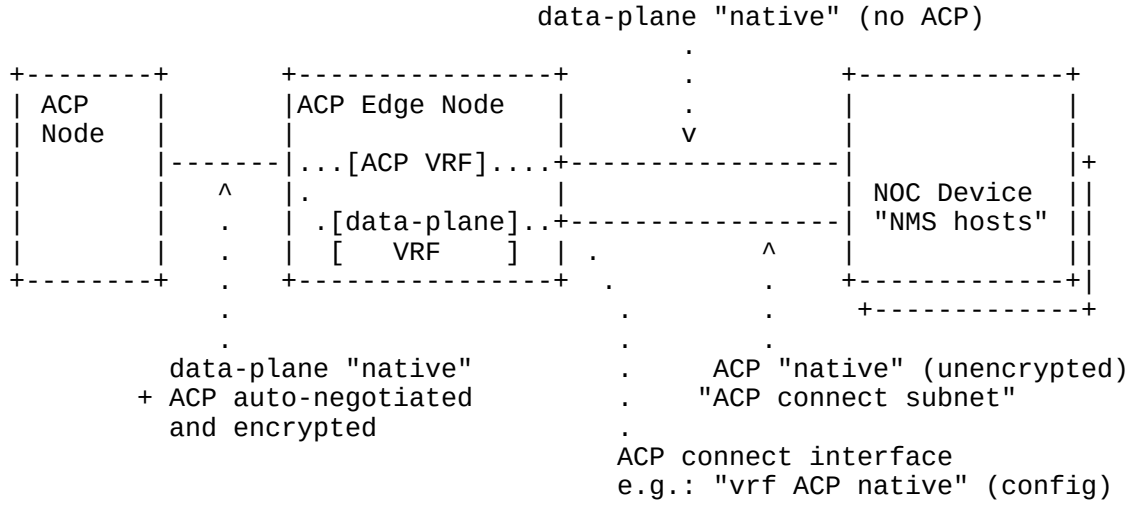


Figure 8: ACP connect

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 must 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.

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 "Autonomic Network Stable Connectivity" [I-D.ietf-anima-stable-connectivity] explains in more detail how the ACP can be integrated in a mixed NOC environment.

The ACP connect interface must be (auto-)configured with an IPv6 address prefix. Is prefix SHOULD be covered by one of the (ULA) prefix(es) used in the ACP. If using non-autonomic configuration, it SHOULD use the ACP Manual Addressing Sub-Scheme (Section 6.10.4). 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] - The NMS host will select the ACP connect interface because any ACP destination address is best 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.

ACP Edge Nodes MUST only forward IPv6 packets received from an ACP connect interface into the ACP that has an IPv6 address from the ACP prefix assigned to this interface (sometimes called "RPF filtering"). This MAY be changed through administrative measures.

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 (such as those physically secure locations like a NOC). For example, the certificate of such node could include an extension required to permit configuration of ACP connect interfaces. This prohibits that a random ACP node with easy physical access that is not meant to run ACP connect could start leaking the ACP when it becomes compromised and the intruder configures ACP connect on it. The full workflow including the mechanism by which a registrar would select which node to give such a certificate to is subject to future work.

8.1.2. Software Components

The ACP connect mechanism be 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 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 to ensure 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 signed software packages. The specification of these mechanisms is subject to future work.

Note that ASA (Autonomic Software Agents) could also be software components as described in this section, but further details of ASAs are subject to future work.

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 autoconfiguration according to [RFC4191].

The ACP edge node acts as the router on the ACP connect subnet, providing the (auto-)configured prefix for the ACP connect subnet to NMS hosts and/or software components. The ACP edge node uses route prefix option of RFC4191 to announce the default route (::/) with a lifetime of 0 and aggregated prefixes for routes in the ACP routing table with normal lifetimes. This will ensure that the ACP edge node does not become a default router, but that the NMS hosts and software components will route the prefixes used in the ACP to the ACP edge node.

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 (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)

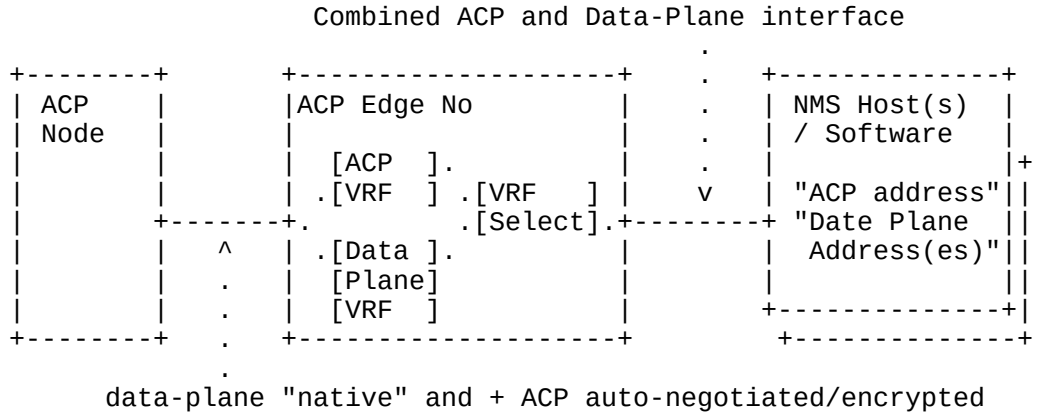


Figure 9: VRF select

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.

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 VRF. This is sometimes called "VRF select". If the ACP VRF has no overlapping IPv6 addresses with the data-plane (as it should), 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.

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 trustworthiness of nodes/software

allowed to participate in the ACP GRASP domain is one of the main reasons why the ACP section describes no solution with non-ACP routers participating in the ACP routing table.

ACP connect interfaces can be dealt with in the GRASP ACP domain like any other ACP interface assuming that any physical ACP connect interface is physically protected from attacks and that the connected Software or NMS Hosts are equally trusted as that on other ACP nodes. ACP edge nodes SHOULD have options to filter GRASP messages in and out of ACP connect interfaces (permit/deny) and MAY have more fine-grained filtering (e.g.: based on IPv6 address of originator or objective).

When using "Combined ACP and Data-Plane Interfaces", care must 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 must 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. ACP through Non-ACP L3 Clouds (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 as follows:

```
remote-peer = [ local-address, method, remote-address ]
local-address  = ip-address
remote-address = transport-address
transport-address =
   [ (ip-address | pattern) ?( , protocol ?(, port)) (, pmtu) ]
ip-address = (ipv4-address | ipv6-address )
method = "IKEv2" / "dTLS" / ..
pattern = some IP address set
```

For each candidate configured remote ACP neighbor, the secure channel protocol "method" is configured with its expected local IP address and remote transport endpoint. Transport protocol and port number for the remote transport endpoint are usually not necessary to configure if defaults for the secure channel protocol method exist.

This is the same information that would be communicated via DULL for L2 adjacent candidate ACP neighbors. DULL is not used because the remote IP address would need to be configured anyhow and if the remote transport address would not be configured but learned via DULL then this would create a third party attack vector.

The secure channel method leverages the configuration to filter incoming connection requests by the remote IP address. This is supplemental security. The primary security is via the mutual domain certificate based authentication of the secure channel protocol.

On a hub node, the remote IP address may be set to some pattern instead of explicit IP addresses. In this case, the node does not attempt to initiate secure channel connections but only acts as their responder. This allows for simple hub&spoke setups for the ACP where some method (subject to further specification) provisions the transport-address of hubs into spokes and hubs accept connections from any spokes. The typical use case for this are spokes connecting via the Internet to hubs. For example, this would be simple extension to BRSKI to allow zero-touch security across the Internet.

Unlike adjacent ACP neighbor connections, configured remote ACP neighbor connections can also be across IPv4. Not all (future) secure channel methods may support running IPv6 (as used in the ACP across the secure channel connection) over IPv4 encapsulation.

Unless the secure channel method supports PMTUD, it needs to be set up with minimum MTU or the path mtu (pmtu) should be configured.

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 to "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.

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.

Future work could envisage an option where the edge nodes of the L3 cloud is configured to automatically forward ACP discovery messages to the right exit point. This optimisation is not considered in this document.

9. Benefits (Informative)

9.1. 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.
- If the domain certificate of an existing ACP node gets revoked, it will automatically be denied access to the ACP as its domain certificate will be validated against a Certificate Revocation List during authentication. Since the revocation check is only done at the establishment of a new security association, existing ones are not automatically torn down. If an immediate disconnect is required, existing sessions to a freshly revoked node can be re-set.

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 certificate revocation list (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, a registrar or Certificate 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 enrolment of new nodes during the partition.

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 trust anchor, a re-merge will be smooth.

Merging two networks with different trust anchors requires the trust anchors to mutually trust each other (for example, by cross-signing). As long as the domain names are different, the addressing will not overlap (see Section 6.10).

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).

9.2. Self-Protection Properties

9.2.1. 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 like 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.

An attacker will not be able to join the ACP unless having a valid domain certificate, also packet injection and sniffing traffic will not be possible due to the security provided by the encryption protocol.

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 fails on some vendor/product/customer limitations. This includes protocols such as SNMP, NTP/PTP, DNS, DHCP, syslog, Radius/Diameter/TACACS, IPFIX/Netflow - just to name a few. 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 AN protocols themselves, either via directed attacks or by denial-of-service attacks. However, as the ACP is built using link-local IPv6 address, remote attacks are impossible. 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.

9.2.2. From the inside

The security model of the ACP is based on trusting all members of the group of nodes that do receive an ACP domain certificate for the same domain. Attacks from the inside by a compromised group member are therefore the biggest challenge.

Group members must overall the secured so that there are no easy way to compromise them, such as data-plane accessible privilege level with simple passwords. This is a lot easier to do in devices whose software is designed from the ground up with security in mind than with legacy software based system where 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).

Further security can be achieved by constraining communication patterns inside the ACP, for example through roles that could be encoded into the domain certificates. This is subject for future work.

9.3. 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 independent of configuration, there is no 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.

While configuration 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 aspect 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.

10. Further Considerations (Informative)

The following sections cover topics that are beyond the primary cope of this document (e.g.: bootstrap), that explain decisions made in this document (e.g.: choice of GRASP) or that explain desirable extensions or implementation details for the ACP that are not considered to be appropriate to standardize in this document.

10.1. BRSKI Bootstrap (ANI)

[I-D.ietf-anima-bootstrapping-keyinfra] (BRSKI) describes how nodes with an IDevID certificate can securely and zero-touch enroll with a domain certificate (LDevID) 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/DS 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 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 registrar (acting as an EST server) must include the subjectAltName / rfc822Name 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 Certificate Authority in an ACP network must not change the subjectAltName / rfc822Name 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 first attempts to perform TLS authentication for BRSKI bootstrap with its expired domain certificate - and only reverts to its IDevID when this fails. This mechanism could also render CRLs unnecessary because the BRSKI registrar in conjunction with the CA would not renew revoked certificates - only a "no-not-renew" list would be necessary on 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 ([I-D.ietf-netconf-zerotouch]) 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.

10.2. 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.

The basic diagnostics is support of (yang) data models representing the complete (auto-)configuration and operational state of all components: BRSKI, GRASP, ACP and the infrastructure used by them: TLS/dTLS, IPsec, certificates, trust anchors, 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. As much as possible, future work should attempt to standardize data models that support common error diagnostic.

- 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 most simple form of diagnostics answering questions like 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 10.3 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, trust anchors.
- 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 authenticate against the trust anchor ?
  - 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 information 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 neighborships.

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:
    + 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).
    + The ACP domain certificate membership check (Section 6.1.2) fails:
      - The neighbors certificate does not have the required trust anchor. Provide diagnostics which trust anchor it has (can identify whom the device belongs to).
      - The neighbors certificate does not have the same domain (or no domain at all). Diagnose domain-name and potentially other other cert info.

- The neighbors certificate has been revoked or could not be authenticated by OCSP.

- The neighbors 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 ?:

o Are there one or more active ACP neighbors with secure channels ?

o Is the RPL routing protocol for the ACP running ?

o Is there a default route to the root in the ACP routing table ?

o Is there for each direct ACP neighbor not reachable over the ACP virtual interface to the root a route in the ACP routing table ?

o Is ACP GRASP running ?

o Is at least one SRV.est objective cached (to support certificate renewal) ?

o Is there at least one BRSKI registrar objective cached (in case BRSKI is supported)

o Is BRSKI proxy operating normally on all interfaces where ACP is operating ?

o ...

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 certificate (IDevID). Only the selected BRSKI-registrar can do this, but it may be difficult to disseminate information about undesired pledges from those 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 considered to be too insecure to make this information available unprotected to all possible neighbors.

An "interested adjacent party" can always determine the IDevID of a BRSKI pledge by behaving like a BRSKI proxy/registrar. Therefore the IDevID 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".

Desirable options for additional diagnostics subject to future work include:

1. Determine if LLDP should be a recommended functionality for ANI devices to improve diagnostics, and if so, which information elements it should signal (insecure).

2. In alternative to LLDP, A DULL GRASP diagnostics objective could be defined to carry these information elements.

3. The IDevID of BRSKI pledges should be included in the selected insecure diagnostics option.

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.

10.3. 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.

10.3.1. 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.

10.3.2. 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).

To provide ACP/ANI resilience against operators configuring interfaces to "down" state, 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 10.3.1.

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.

10.3.2.1. 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 into the security of ACP/ANI operations need 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 a 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.

10.3.2.2. 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 (autonomous) 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 autoconfigure a BFD 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 10.2 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.

10.3.2.3. 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 autoconfiguring 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 diagnostics that automatically terminates in a predetermined manner without dependence on external input ensuring the link will become operational again.

10.3.2.4. Power Consumption

Power consumption of "physical down" interfaces may be significantly lower than those in "admin down" state, for example on long range fiber interfaces. Assuming reasonable clocks on devices, mechanisms for infrequent periodic probing could allow to automatically establish ACP connectivity across such links. Bring up interfaces for 5 seconds to probe if there is an ACP neighbor on the remote end every 500 seconds = 1% power consumption.

10.3.3. 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".

10.3.4. Which interfaces to auto-enable ?

(Section 6.3) 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.

10.3.5. 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 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".

10.3.5.1. 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 most simple 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 he likely never even heard of it could be quite irritating to him. Especially when "down" behavior is changed to "admin down".

Automatically setting "ANI enable" on brownfield nodes where the operator is unaware of it could also be a critical security issue depending on the vouchers used by BRKSI on these nodes. An attacker could claim to be the owner of these devices and create an ACP that the attacker has access/control over. In network where the operator explicitly wants to enable the ANI this could not happen, because he would create a BRSKI registrar that would discover attack attempts. 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.

10.3.5.2. Greenfield nodes

A "greenfield" node is one that did not have any prior configuration.

For greenfield nodes, only "ANI enable" is relevant. If another mechanism than BRSKI is used to (zero-touch) bootstrap a node, then it is up to that mechanism to provision domain certificates and to set global "ACP enable" as desired.

Nodes supporting full ANI functionality set "ANI enable" automatically when they decide that they are greenfield, e.g.: that they are powering on from factory condition. They will then put all native interfaces into "admin down" state and start to perform BRSKI pledge functionality - and once a domain certificate is enrolled they automatically enable ACP.

Attempts for BRSKI pledge operations in greenfield state should terminate automatically when another method of configuring the node is used. Methods that indicate some form of physical possession of the device such as configuration via the serial console could lead to immediate termination of BRSKI, while other parallel autoconfiguration methods subject to remote attacks might lead to BRSKI termination only after they were successful. Details of this may vary widely over different type of nodes. When BRSKI pledge operation terminates, this will automatically unset "ANI enable" and should terminate any temporarily needed state on the device to perform BRSKI - DULL GRASP, BRSKI pledge and any IPv6 configuration on interfaces.

10.3.6. 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 property in the certificate (e.g.: in the domain information extension field) subject to future work: 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.

10.3.7. 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.

10.4. 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.

10.4.1. LLDP

LLDP (and Cisco’s similar 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.

10.4.2. mDNS and L2 support

mDNS [RFC6762] with DNS-SD RRs (Resource Records) 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.

10.4.3. 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.

10.5. Choice of routing protocol (RPL)

This Appendix explains 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. RPL also has other scalability improvements, such as selecting only a subset of peers instead of all possible ones, and trickle support for information synchronization.

- 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 destination-oriented directed acyclic graph (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.

- o No need for path optimisation: 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 optimisation schemes become necessary in the future, but RPL can be expanded (see point "Extensibility" above).

10.6. Extending ACP channel negotiation (via GRASP)

The mechanism described in the normative part of this document to support multiple different ACP secure channel protocols without a single network wide MTI protocol is important to allow extending secure ACP channel protocols beyond what is specified in this document, but it will run into problem if it would be used for multiple protocols:

The need to potentially have multiple of these security associations even temporarily run in parallel to determine which of them works best does not support the most lightweight implementation options.

The simple policy of letting one side (Alice) decide what is best may not lead to the mutual best result.

The two limitations can easier be solved if the solution was more modular and as few as possible initial secure channel negotiation protocols would be used, and these protocols would then take on the responsibility to support more flexible objectives to negotiate the mutually preferred ACP security channel protocol.

IKEv2 is the IETF standard protocol to negotiate network security associations. It is meant to be extensible, but it is unclear whether it would be feasible to extend IKEv2 to support possible future requirements for ACP secure channel negotiation:

Consider the simple case where the use of native IPsec vs. IPsec via GRE is to be negotiated and the objective is the maximum throughput. Both sides would indicate some agreed upon performance metric and the preferred encapsulation is the one with the higher performance of the slower side. IKEv2 does not support negotiation with this objective.

Consider dTLS and some form of 802.1AE ([MACSEC]) are to be added as negotiation options - and the performance objective should work across all IPsec, dDTLS and 802.1AE options. In the case of MacSEC, the negotiation would also need to determine a key for the peering. It is unclear if it would be even appropriate to consider extending the scope of negotiation in IKEv2 to those cases. Even if feasible to define, it is unclear if implementations of IKEv2 would be eager to adopt those type of extension given the long cycles of security

testing that necessarily goes along with core security protocols such as IKEv2 implementations.

A more modular alternative to extending IKEv2 could be to layer a modular negotiation mechanism on top of the multitude of existing or possible future secure channel protocols. For this, GRASP over TLS could be considered as a first ACP secure channel negotiation protocol. The following are initial considerations for such an approach. A full specification is subject to a separate document:

To explicitly allow negotiation of the ACP channel protocol, GRASP over a TLS connection using the GRASP_LISTEN_PORT and the nodes and peers link-local IPv6 address is used. When Alice and Bob support GRASP negotiation, they do prefer it over any other non-explicitly negotiated security association protocol and should wait trying any non-negotiated ACP channel protocol until after it is clear that GRASP/TLS will not work to the peer.

When Alice and Bob successfully establish the GRASP/TSL session, they will negotiate the channel mechanism to use using objectives such as performance and perceived quality of the security. After agreeing on a channel mechanism, Alice and Bob start the selected Channel protocol. Once the secure channel protocol is successfully running, the GRASP/TLS connection can be kept alive or timed out as long as the selected channel protocol has a secure association between Alice and Bob. When it terminates, it needs to be re-negotiated via GRASP/TLS.

Notes:

- Negotiation of a channel type may require IANA assignments of code points.

- TLS is subject to reset attacks, which IKEv2 is not. Normally, ACP connections (as specified in this document) will be over link-local addresses so the attack surface for this one issue in TCP should be reduced (note that this may not be true when ACP is tunneled as described in Section 8.2.2.

- GRASP packets received inside a TLS connection established for GRASP/TLS ACP negotiation are assigned to a separate GRASP domain unique to that TLS connection.

10.7. 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 domain information field of 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 using certificates from the same CA using the same domain field. GRASP inside the ACP is run across all transitively connected ACP nodes in a domain.

The rsub element in the domain information field primarily allows to use addresses from different ULA prefixes. One use case is to create multiple networks that initially may be separated, but where it should be possible to connect them without further extensions to ACP when necessary.

Another use case for routing subdomains is as the starting point for structuring routing inside an ACP. For example, different routing subdomains could run different routing protocols or different instances of RPL and auto-aggregation / distribution of routes could be done across inter routing subdomain ACP channels based on negotiation (e.g.: via GRASP). This is subject for further work.

RPL scales very well. It is not necessary to use multiple routing subdomains to scale ACP domains in a way it would be possible 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 domain information field. 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 domain information element 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".

Future extensions, primarily through intent can create more flexible options how to build ACP domains.

Intent could modify the ACP connection check to permit connections between different domains.

If different domains use the same CA one would change the ACP setup to permit for the ACP to be established between the two ACP nodes, but no routing nor ACP GRASP to be built across this adjacency. The main difference over routing subdomains is to not permit for the ACP GRASP instance to be built across the adjacency. Instead, one would only build a point to point GRASP instance between those peers to negotiate what type of exchanges are desired across that connection. This would include routing negotiation, how much GRASP information to transit and what data-plane forwarding should be done. This approach could also allow for Intent to only be injected into the network from one side and propagate via this GRASP connection.

If different domains have different CAs, they should start to trust each other by intent injected into both domains that would add the other domains CA as a trust point during the ACP connection setup - and then following up with the previous point of inter-domain connections across domains with the same CA (e.g.: GRASP negotiation).

10.8. 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 /50 IPv6 prefix. This prefix could simply be a configuration of the registrars when using BRSKI to enroll the domain certificates - instead of the registrar deriving the /50 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 empty/unused. 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 Alice and who is Bob 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].

The routing protocol chosen by the ACP design (RPL) does explicitly not optimize for shortest paths and fastest convergence. Variations of the ACP may want to use a different routing protocol.

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 enrolment, still leveraging the BRSKI TLS connection and therefore not introducing any additional connectivity requirements.

Last but not least, secure channel protocols including their encapsulation are easily added to ACP solutions. Secure channels may even be replaced by simple neighbor authentication to create simplified ACP variations for environments where no real security is required but just protection against non-malicious misconfiguration. Or for environments where all traffic is known or forced to be end-to-end protected and other means for infrastructure protection are used. 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.

11. Security Considerations

An ACP is self-protecting and there is no need to apply configuration to make it secure. Its security therefore does not depend on configuration.

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.

- Security can be compromised by implementation errors (bugs), as in all products.

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.

Fundamentally, security depends on correct operation, implementation and architecture. Autonomic approaches such as the ACP largely eliminate the dependency on correct operation; implementation and architectural mistakes are still possible, as in all networking technologies.

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 nodes domain certificate as described in Section 6.1.1. This allows even verification of ownership of a peers 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 6.8.2.

ACP provides for secure, resilient zero-touch discovery of EST servers for certificate renewal. See Section 6.1.3.

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 6.5 and Section 10.6.

The ACP is designed to minimize attacks from the outside by minimizing its dependency against any non-ACP operations on a node. The only dependency in the specification in this document is the need to share link-local addresses for the ACP secure channel encapsulation with the data-plane. See Section 6.12.2.

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 Section 10.1.

12. IANA Considerations

This document defines the "Autonomic Control Plane".

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 6.3.

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 6.1.3.

Note that the objective format "SRV.<service-name>" is intended to be used for any <service-name> that is an [RFC6335] registered service name. This is a proposed update to the GRASP registry subject to future work and only mentioned here for informational purposed to explain the unique format of the objective name.

The IANA is requested to create an ACP Parameter Registry with currently one registry table - the "ACP Address Type" table.

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 Figure 4) / ACP Manual Addressing Sub-Scheme (ACP RFC Section 6.10.4)
1: ACP Vlong Addressing Sub-Scheme (ACP RFC Section 6.10.5)

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 and Sheng Jiang for their thorough reviews and to Pascal Thubert and Michael Richardson to provide the details for the recommendations of the use of RPL in the ACP

Further input and suggestions were received from: Rene Struik, Brian Carpenter, Benoit Claise.

14. Change log [RFC Editor: Please remove]

14.1. Initial version

First version of this document: draft-behringer-autonomic-control-plane

14.2. draft-behringer-anima-autonomic-control-plane-00

Initial version of the anima document; only minor edits.

14.3. draft-behringer-anima-autonomic-control-plane-01

- Clarified that the ACP should be based on, and support only IPv6.
- Clarified in intro that ACP is for both, between devices, as well as for access from a central entity, such as an NMS.
- Added a section on how to connect an NMS system.
- Clarified the hop-by-hop crypto nature of the ACP.
- Added several references to GDNP as a candidate protocol.
- Added a discussion on network split and merge. Although, this should probably go into the certificate management story longer term.

14.4. draft-behringer-anima-autonomic-control-plane-02

Addresses (numerous) comments from Brian Carpenter. See mailing list for details. The most important changes are:

- Introduced a new section "overview", to ease the understanding of the approach.
- Merged the previous "problem statement" and "use case" sections into a mostly re-written "use cases" section, since they were overlapping.
- Clarified the relationship with draft-ietf-anima-stable-connectivity

14.5. draft-behringer-anima-autonomic-control-plane-03

- Took out requirement for IPv6 --> that’s in the reference doc.
- Added requirement section.
- Changed focus: more focus on autonomic functions, not only virtual out of band. This goes a bit throughout the document, starting with a changed abstract and intro.

14.6. draft-ietf-anima-autonomic-control-plane-00

No changes; re-submitted as WG document.

14.7. draft-ietf-anima-autonomic-control-plane-01

- Added some paragraphs in addressing section on "why IPv6 only", to reflect the discussion on the list.
- Moved the data-plane ACP out of the main document, into an appendix. The focus is now the virtually separated ACP, since it has significant advantages, and isn’t much harder to do.
- Changed the self-creation algorithm: Part of the initial steps go into the reference document. This document now assumes an adjacency table, and domain certificate. How those get onto the device is outside scope for this document.
- Created a new section 6 "workarounds for non-autonomic nodes", and put the previous controller section (5.9) into this new section. Now, section 5 is "autonomic only", and section 6 explains what to do with non-autonomic stuff. Much cleaner now.

- Added an appendix explaining the choice of RPL as a routing protocol.

- Formalised the creation process a bit more. Now, we create a "candidate peer list" from the adjacency table, and form the ACP with those candidates. Also it explains now better that policy (Intent) can influence the peer selection. (section 4 and 5)

- Introduce a section for the capability negotiation protocol (section 7). This needs to be worked out in more detail. This will likely be based on GRASP.

- Introduce a new parameter: ACP tunnel type. And defines it in the IANA considerations section. Suggest GRE protected with IPSec transport mode as the default tunnel type.

- Updated links, lots of small edits.

14.8. draft-ietf-anima-autonomic-control-plane-02

- Added explicitly text for the ACP channel negotiation.

- Merged draft-behringer-anima-autonomic-addressing-02 into this document, as suggested by WG chairs.

14.9. draft-ietf-anima-autonomic-control-plane-03

- Changed Neighbor discovery protocol from GRASP to mDNS. Bootstrap protocol team decided to go with mDNS to discover bootstrap proxy, and ACP should be consistent with this. Reasons to go with mDNS in bootstrap were a) Bootstrap should be reuseable also outside of full anima solutions and introduce as few as possible new elements. mDNS was considered well-known and very-likely even pre-existing in low-end devices (IoT). b) Using GRASP both for the insecure neighbor discovery and secure ACP operatations raises the risk of introducing security issues through implementation issues/ non-isolation between those two instances of GRASP.

- Shortened the section on GRASP instances, because with mDNS being used for discovery, there is no insecure GRASP session any longer, simplifying the GRASP considerations.

- Added certificate requirements for ANIMA in section 5.1.1, specifically how the ANIMA information is encoded in subjectAltName.

- Deleted the appendix on "ACP without separation", as originally planned, and the paragraph in the main text referring to it.

- Deleted one sub-addressing scheme, focusing on a single scheme now.
- Included information on how ANIMA information must be encoded in the domain certificate in section "preconditions".
- Editorial changes, updated draft references, etc.

14.10. draft-ietf-anima-autonomic-control-plane-04

Changed discovery of ACP neighbor back from mDNS to GRASP after revisiting the L2 problem. Described problem in discovery section itself to justify. Added text to explain how ACP discovery relates to BRSKY (bootstrap) discovery and pointed to Michael Richardsons draft detailing it. Removed appendix section that contained the original explanations why GRASP would be useful (current text is meant to be better).

14.11. draft-ietf-anima-autonomic-control-plane-05

- Section 5.3 (candidate ACP neighbor selection): Add that Intent can override only AFTER an initial default ACP establishment.
- Section 6.10.1 (addressing): State that addresses in the ACP are permanent, and do not support temporary addresses as defined in RFC4941.
- Modified Section 6.3 to point to the GRASP objective defined in draft-carpenter-anima-ani-objectives. (and added that reference)
- Section 6.10.2: changed from MD5 for calculating the first 40 bits to SHA256; reason is MD5 should not be used any more.
- Added address sub-scheme to the IANA section.
- Made the routing section more prescriptive.
- Clarified in Section 8.1.1 the ACP Connect port, and defined that term "ACP Connect".
- Section 8.2: Added some thoughts (from mcr) on how traversing a L3 cloud could be automated.
- Added a CRL check in Section 6.7.
- Added a note on the possibility of source-address spoofing into the security considerations section.

- Other editoral changes, including those proposed by Michael Richardson on 30 Nov 2016 (see ANIMA list).

14.12. draft-ietf-anima-autonomic-control-plane-06

- Added proposed RPL profile.
- detailed dTLS profile - dTLS with any additional negotiation/ signaling channel.
- Fixed up text for ACP/GRE encap. Removed text claiming its incompatible with non-GRE IPsec and detailled it.
- Added text to suggest admin down interfaces should still run ACP.

14.13. draft-ietf-anima-autonomic-control-plane-07

- Changed author association.
- Improved ACP connect setion (after confusion about term came up in the stable connectivity draft review). Added picture, defined complete terminology.
- Moved ACP channel negotiation from normative section to appendix because it can in the timeline of this document not be fully specified to be implementable. Aka: work for future document. That work would also need to include analysing IKEv2 and describin the difference of a proposed GRASP/TLS solution to it.
- Removed IANA request to allocate registry for GRASP/TLS. This would come with future draft (see above).
- Gave the name "ACP information field" to the field in the certificate carrying the ACP address and domain name.
- Changed the rules for mutual authentication of certificates to rely on the domain in the ACP information field of the certificate instead of the OU in the certificate. Also renewed the text pointing out that the ACP information field in the certificate is meant to be in a form that it does not disturb other uses of the certificate. As long as the ACP expected to rely on a common OU across all certificates in a domain, this was not really true: Other uses of the certificates might require different OUs for different areas/type of devices. With the rules in this draft version, the ACP authentication does not rely on any other fields in the certificate.

- Added an extension field to the ACP information field so that in the future additional fields like a subdomain could be inserted. An example using such a subdomain field was added to the pre-existing text suggesting sub-domains. This approach is necessary so that there can be a single (main) domain in the ACP information field, because that is used for mutual authentication of the certificate. Also clarified that only the register(s) SHOULD/MUST use that the ACP address was generated from the domain name - so that we can easier extend change this in extensions.

- Took the text for the GRASP discovery of ACP neighbors from Brians grasp-ani-objectives draft. Alas, that draft was behind the latest GRASP draft, so i had to overhaul. The mayor change is to describe in the ACP draft the whole format of the M_FLOOD message (and not only the actual objective). This should make it a lot easier to read (without having to go back and forth to the GRASP RFC/draft). It was also necessary because the locator in the M_FLOOD messages has an important role and its not coded inside the objective. The specification of how to format the M_FLOOD message shuold now be complete, the text may be some duplicate with the DULL specificateion in GRASP, but no contradiction.

- One of the main outcomes of reworking the GRASP section was the notion that GRASP announces both the candidate peers IPv6 link local address but also the support ACP security protocol including the port it is running on. In the past we shied away from using this information because it is not secured, but i think the additional attack vectors possible by using this information are negligible: If an attacker on an L2 subnet can fake another devices GRASP message then it can already provide a similar amount of attack by purely faking the link-local address.

- Removed the section on discovery and BRSKI. This can be revived in the BRSKI document, but it seems mood given how we did remove mDNS from the latest BRSKI document (aka: this section discussed discrepancies between GRASP and mDNS discovery which should not exist anymore with latest BRSKI.

- Tried to resolve the EDNOTE about CRL vs. OCSP by pointing out we do not specify which one is to be used but that the ACP should be used to reach the URL included in the certificate to get to the CRL storage or OCSP server.

- Changed ACP via IPsec to ACP via IKEv2 and restructured the sections to make IPsec native and IPsec via GRE subsections.

- No need for any assigned dTLS port if ACP is run across dTLS because it is signaled via GRASP.

14.14. draft-ietf-anima-autonomic-control-plane-08

Modified mentioning of BRSKI to make it consistent with current (07/2017) target for BRSKI: MASA and IDevID are mandatory. Devices with only insecure UDI would need a security reduced variant of BRSKI. Also added mentioning of Netconf Zero-Touch. Made BRSKI non-normative for ACP because wrt. ACP it is just one option how the domain certificate can be provisioned. Instead, BRSKI is mandatory when a device implements ANI which is ACP+BRSKI.

Enhanced text for ACP across tunnels to decribe two options: one across configured tunnels (GRE, IPinIP etc) a more efficient one via directed DULL.

Moved decription of BRSKI to appendex to emphasize that BRSKI is not a (normative) dependency of GRASP, enhanced text to indicate other options how Domain Certificates can be provisioned.

Added terminology section.

Separated references into normative and non-normative.

Enhanced section about ACP via "tunnels". Defined an option to run ACP secure channel without an outer tunnel, discussed PMTU, benefits of tunneling, potential of using this with BRSKI, made ACP via GREP a SHOULD requirement.

Moved appendix sections up before IANA section because there where concerns about appendices to be to far on the bottom to be read. Added (Informative) / (Normative) to section titles to clarify which sections are informative and which are normative

Moved explanation of ACP with L2 from precondition to separate section before workarounds, made it instructive enough to explain how to implement ACP on L2 ports for L3/L2 switches and made this part of normative requirement (L2/L3 switches SHOULD support this).

Rewrote section "GRASP in the ACP" to define GRASP in ACP as mandatory (and why), and define the ACP as security and transport substrate to GRASP in ACP. And how it works.

Enhanced "self-protection" properties section: protect legacy management protocols. Security in ACP is for protection from outside and those legacy protocols. Otherwise need end-to-end encryption also inside ACP, e.g.: with domain certificate.

Enhanced initial domain certificate section to include requirements for maintenance (renewal/revocation) of certificates. Added

explanation to BRSKI informative section how to handle very short lived certificates (renewal via BRSKI with expired cert).

Modified the encoding of the ACP address to better fit RFC822 simple local-parts (":" as required by RFC5952 are not permitted in simple dot-atoms according to RFC5322. Removed reference to RFC5952 as its now not needed anymore.

Introduced a sub-domain field in the ACP information in the certificate to allow defining such subdomains with depending on future Intent definitions. It also makes it clear what the "main domain" is. Scheme is called "routing subdomain" to have a unique name.

Added V8 (now called Vlong) addressing sub-scheme according to suggestion from mcr in his mail from 30 Nov 2016 (https://mailarchive.ietf.org/arch/msg/anima/nZpEphrTqDCBdzsKMpaIn2gsIzI). Also modified the explanation of the single V bit in the first sub-scheme now renamed to Zone sub-scheme to distinguish it.

14.15. draft-ietf-anima-autonomic-control-plane-09

Added reference to RFC4191 and explained how it should be used on ACP edge routers to allow autoconfiguration of routing by NMS hosts. This came after review of stable connectivity draft where ACP connect is being referred to.

V8 addressing Sub-Scheme was modified to allow not only /8 device-local address space but also /16. This was in response to the possible need to have maybe as much as 2^12 local addresses for future encaps in BRSKI like IPinIP. It also would allow fully autonomic address assignment for ACP connect interfaces from this local address space (on an ACP edge device), subject to approval of the implied update to rfc4291/rfc4193 (IID length). Changed name to Vlong addressing sub-scheme.

Added text in response to Brian Carpenters review of draft-ietf-anima-stable-connectivity-04.

- The stable connectivity draft was vaguely describing ACP connect behavior that is better standardized in this ACP draft.
- Added new ACP "Manual" addressing sub-scheme with /64 subnets for use with ACP connect interfaces. Being covered by the ACP ULA prefix, these subnets do not require additional routing entries for NMS hosts. They also are fully 64-bit IID length compliant and therefore not subject to 4191bis considerations. And they

avoid that operators manually assign prefixes from the ACP ULA prefixes that might later be assigned autonomiously.

- ACP connect auto-configuration: Defined that ACP edge devices, NMS hosts should use RFC4191 to automatically learn ACP prefixes. This is especially necessary when the ACP uses multiple ULA prefixes (via e.g.: the rsub domain certificate option), or if ACP connect subinterfaces use manually configured prefixes NOT covered by the ACP ULA prefixes.

- Explained how rfc6724 is (only) sufficient when the NMS host has a separate ACP connect and data-plane interface. But not when there is a single interface.

- Added a separate subsection to talk about "software" instead of "NMS hosts" connecting to the ACP via the "ACP connect" method. The reason is to point out that the "ACP connect" method is not only a workaround (for NMS hosts), but an actual desirable long term architectural component to modularily build software (e.g.: ASA or OAM for VNF) into ACP devices.

- Added a section to define how to run ACP connect across the same interface as the data-plane. This turns out to be quite challenging because we only want to rely on existing standards for the network stack in the NMS host/software and only define what features the ACP edge device needs.

- Added section about use of GRASP over ACP connect.

- Added text to indicate packet processing/filtering for security: filter incorrect packets arriving on ACP connect interfaces, diagnose on RPL root packets to incorrect destination address (not in ACP connect section, but because of it).

- Reaffirm security goal of ACP: Do not permit non-ACP routers into ACP routing domain.

Made this ACP document be an update to RFC4291 and RFC4193. At the core, some of the ACP addressing sub-schemes do effectively not use 64-bit IIDs as required by RFC4191 and debated in rfc4191bis. During 6man in prague, it was suggested that all documents that do not do this should be classified as such updates. Add a rather long section that summarizes the relevant parts of ACP addressing and usage and. Aka: This section is meant to be the primary review section for readers interested in these changes (e.g.: 6man WG.).

Added changes from Michael Richardsons review https://github.com/anima-wg/autonomic-control-plane/pull/3/commits, textual and:

- ACP discovery inside ACP is bad *doh*!.
- Better CA trust and revocation sentences.
- More details about RPL behavior in ACP.
- black hole route to avoid loops in RPL.

Added requirement to terminate ACP channels upon cert expiry/ revocation.

Added fixes from 08-mcr-review-reply.txt (on github):

- AN Domain Names are FQDNs.
- Fixed bit length of schemes, numerical writing of bits (00b/01b).
- Lets use US american english.

14.16. draft-ietf-anima-autonomic-control-plane-10

Used the term routing subdomain more consistently where previously only subdomain was used. Clarified use of routing subdomain in creation of ULA "global ID" addressing prefix.

6.7.1.* Changed native IPsec encapsulation to tunnel mode (necessary), explaned why. Added notion that ESP is used, added explanations why tunnel/transport mode in native vs. GRE cases.

6.10.3/6.10.5 Added term "ACP address range/set" to be able to better explain how the address in the ACP certificate is actually the base address (lowest address) of a range/set that is available to the device.

6.10.4 Added note that manual address sub-scheme addresses must not be used within domain certificates (only for explicit configuration).

6.12.5 Refined explanation of how ACP virtual interfaces work (p2p and multipoint). Did seek for pre-existing RFCs that explain how to built a multi-access interface on top of a full mesh of p2p connections (6man WG, anima WG mailing lists), but could not find any prior work that had a succinct explanation. So wrote up an explanation here. Added hopefully all necessary and sufficient details how to map ACP unicast packets to ACP secure channel, how to deal with ND packet details. Added verbage for ACP not to assign the virtual interface link-local address from the underlying interface. Addd note that GRAP link-local messages are treated specially but logically the same. Added paragraph about NBMA interfaces.

remaining changes from Brian Carpenters review. See Github file draft-ietf-anima-autonomic-control-plane/08-carpenter-review-reply.tx for more detailst:

Added multiple new RFC references for terms/technologies used.

Fixed verbage in several places.

2. (terminology) Added 802.1AR as reference.

2. Fixed up definition of ULA.

6.1.1 Changed definition of ACP information in cert into ABNF format. Added warning about maximum size of ACP address field due to domain-name limitations.

6.2 Mentioned API requirement between ACP and clients leveraging adjacency table.

6.3 Fixed TTL in GRASP example: msec, not hop-count!.

6.8.2 MAYOR: expanded security/transport substrate text:

Introduced term ACP GRASP virtual interface to explain how GRASP link-local multicast messages are encapsulated and replicated to neighbors. Explain how ACP knows when to use TLS vs. TCP (TCP only for link-local address (sockets). Introduced "ladder" picture to visualize stack.

6.8.2.1 Expanded discussion/explanation of security model. TLS for GRASP unicsast connections across ACP is double encryption (plus underlying ACP secure channel), but highly necessary to avoid very simple man-in-the-middle attacks by compromised ACP members on-path. Ultimately, this is done to ensure that any apps using GRASP can get full end-to-end secrecy for information sent across GRASP. But for publically known ASA services, even this will not provide 100% security (this is discussed). Also why double encryption is the better/easier solution than trying to optimize this.

6.10.1 Added discussion about pseudo-random addressing, scanning-attaacks (not an issue for ACP).

6.12.2 New performance requirements section added.

6.10.1 Added notion to first experiment with existing addressing schemes before defining new ones - we should be flexible enough.

6.3/7.2 clarified the interactions between MLD and DULL GRASP and specified what needs to be done (e.g.: in 2 switches doing ACP per L2 port).

12. Added explanations and cross-references to various security aspects of ACP discussed elsewhere in the document.

13. Added IANA requirements.

Added RFC2119 boilerplate.

14.17. draft-ietf-anima-autonomic-control-plane-11

Same text as -10 Unfortunately when uploading -10 .xml/.txt to datatracker, a wrong version of .txt got uploaded, only the .xml was correct. This impacts the -10 html version on datatra cker and the PDF versions as well. Because rfcdiff also compares the .txt version, this -11 version was crea ted so that one can compare changes from -09 and changes to the next version (-12).

14.18. draft-ietf-anima-autonomic-control-plane-12

Sheng Jiangs extensive review. Thanks! See Github file draft-ietf-anima-autonomic-control-plane/09-sheng-review-reply.txt for more details. Many of the larger changes listed below where inspired by the review.

Removed the claim that the document is updating RFC4291,RFC4193 and the section detailling it. Done on suggestion of Michael Richardson - just try to describe use of addressing in a way that would not suggest a need claim update to architecture.

Terminology cleanup:

- Replaced "device" with "node" in text. Kept "device" only when referring to "physical node". Added definitions for those words. Includes changes of derived terms, especially in addressing: "Node-ID" and "Node-Number" in the addressing details.

- Replaced term "autonomic FOOBAR" with "acp FOOBAR" as whereever appropriate: "autonomic" would imply that the node would need to support more than the ACP, but that is not correct in most of the cases. Wanted to make sure that implementers know they only need to support/implement ACP - unless stated otherwise. Includes "AN->ACP node", "AN->ACP adjacency table" and so on.

1 Added explanation in the introduction about relationship between ACP, BRSKI, ANI and Autonomic Networks.

6.1.1 Improved terminology and features of the certificate information field. Now called domain information field instead of ACP information field. The acp-address field in the domain information field is now optional, enabling easier introduction of various future options.

6.1.2 Moved ACP domainer membership check from section 6.6 to (ACP secure channels setup) here because it is not only used for ACP secure channel setup.

6.1.3 Fix text about certificate renewal after discussion with Max Pritikin/Michael Richardson/Brian Carpenter:

- Version 10 erroneously assumed that the certificate itself could store a URL for renewal, but that is only possible for CRL URLs. Text now only refers to "remembered EST server" without implying that this is stored in the certificate.

- Objective for RFC7030/EST domain certificate renewal was changed to "SRV.est" See also IANA section for explanation.

- Removed detail of distance based service selection. This can be better done in future work because it would require a lot more detail for a good DNS-SD compatible approach.

- Removed detail about trying to create more security by using ACP address from certificate of peer. After rethinking, this does not seem to buy additional security.

6.10 Added reference to 6.12.5 in initial use of "loopback interface" in section 6.10 in result of email discussion michaelR/michaelB.

10.2 Introduced informational section (diagnostics) because of operational experience - ACP/ANI undeployable without at least diagnostics like this.

10.3 Introduced informational section (enabling/disabling) ACP. Important to discuss this for security reasons (e.g.: why to never never auto-enable ANI on brownfield devices), for implementers and to answer ongoing questions during WG meetings about how to deal with shutdown interface.

10.8 Added informational section discussing possible future variations of the ACP for potential adopters that cannot directly use the complete solution described in this document unmodified.

15. References

15.1. 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.

[RFC1034] Mockapetris, P., "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>.

[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>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[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>.

[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>.

[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>.

[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>.

15.2. Informative References

[AR8021] IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", December 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

[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-07 (work in progress), July 2017.

[I-D.ietf-anima-prefix-management] Jiang, S., Du, Z., Carpenter, B., and Q. Sun, "Autonomic IPv6 Edge Prefix Management in Large-scale Networks", draft-ietf-anima-prefix-management-05 (work in progress), August 2017.

[I-D.ietf-anima-reference-model] Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., Pierre, P., Liu, B., Nobre, J., and J. Strassner, "A Reference Model for Autonomic Networking", draft-ietf-anima-reference-model-04 (work in progress), July 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-06 (work in progress), September 2017.

[I-D.ietf-netconf-zerotouch] Watsen, K., Abrahamsson, M., and I. Farrer, "Zero Touch Provisioning for NETCONF or RESTCONF based Management", draft-ietf-netconf-zerotouch-17 (work in progress), September 2017.

[I-D.ietf-roll-useofrplinfo] Robles, I., Richardson, M., and P. Thubert, "When to use RFC 6553, 6554 and IPv6-in-IPv6", draft-ietf-roll-useofrplinfo-16 (work in progress), July 2017.

[MACSEC] IEEE SA-Standards Board, "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., "Host extensions for IP multicasting", STD 5, RFC 1112, DOI 10.17487/RFC1112, August 1989, <https://www.rfc-editor.org/info/rfc1112>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., 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>.

[RFC2821] Klensin, J., Ed., "Simple Mail Transfer Protocol", RFC 2821, DOI 10.17487/RFC2821, April 2001, <https://www.rfc-editor.org/info/rfc2821>.

[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>.

[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>.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <https://www.rfc-editor.org/info/rfc5321>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

Authors' Addresses

Michael H. Behringer (editor)

Email: michael.h.behringer@gmail.com

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

Email: tte+ietf@cs.fau.de

Steinthor Bjarnason
Arbor Networks
2727 South State Street, Suite 200
Ann Arbor MI 48104
United States

Email: sbjarnason@arbor.net

ANIMA WG
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

M. Pritikin
Cisco
M. Richardson
SSW
M. Behringer
Cisco
S. Bjarnason
Arbor Networks
K. Watsen
Juniper Networks
October 30, 2017

Bootstrapping Remote Secure Key Infrastructures (BRSKI)
draft-ietf-anima-bootstrapping-keyinfra-09

Abstract

This document specifies automated bootstrapping of a remote secure key infrastructure (BRSKI) using vendor installed X.509 certificate, in combination with a vendor's authorizing service, both online and offline. Bootstrapping a new device can occur using a routable address and a cloud service, or using only link-local connectivity, or on limited/disconnected networks. Support for lower security models, including devices with minimal identity, is described for legacy reasons but not encouraged. Bootstrapping is complete when the cryptographic identity of the new key infrastructure is successfully deployed to the device but the established secure connection can be used to deploy a locally issued certificate to the device as well.

Table of Contents

1. Introduction

BRSKI provides a foundation to securely answer the following questions between an element of the network domain called the "Registrar" and an unconfigured and untouched device called a "Pledge":

- Registrar authenticating the Pledge: "Who is this device? What is its identity?"
- Registrar authorization the Pledge: "Is it mine? Do I want it? What are the chances it has been compromised?"
- Pledge authenticating the Registrar/Domain: "What is this domain's identity?"
- Pledge authorization the Registrar: "Should I join it?"

This document details protocols and messages to the endpoints to answer the above questions. The Registrar actions derive from Pledge identity, third party cloud service communications, and local access control lists. The Pledge actions derive from a cryptographically protected "voucher" message delivered through the Registrar but originating at a Manufacturer Authorized Signing Authority.

The syntactic details of vouchers are described in detail in [I-D.ietf-anima-voucher]. This document details automated protocol mechanisms to obtain vouchers, including the definition of a 'voucher-request' message that is a minor extension to the voucher format (see Section 3).

BRSKI results in the Pledge storing an X.509 root certificate sufficient for verifying the Registrar identity. In the process a TLS connection is established which can be directly used for Enrollment over Secure Transport (EST). In effect BRSKI provides an automated mechanism for the "Bootstrap Distribution of CA Certificates" described in [RFC7030] Section 4.1.1 wherein the Pledge "MUST [...]. engage a human user to authorize the CA certificate using out-of-band" information". With BRSKI the Pledge now can automate this process using the voucher. Integration with a complete EST enrollment is optional but trivial.

BRSKI is agile enough to support bootstrapping alternative key infrastructures, such as a symmetric key solutions, but no such system is described in this document.

1.1. Other Bootstrapping Approaches

To literally "pull yourself up by the bootstraps" is an impossible action. Similarly the secure establishment of a key infrastructure without external help is also an impossibility. Today it is commonly accepted that the initial connections between nodes are insecure, until key distribution is complete, or that domain-specific keying material is pre-provisioned on each new device in a costly and non-scalable manner. Existing mechanisms are known as non-secured 'Trust on First Use' (TOFU) [RFC7435], 'resurrecting duckling' [Stajano99theresurrecting] or 'pre-staging'.

Another approach is to try and minimize user actions during bootstrapping. The enrollment protocol EST [RFC7030] details a set of non-autonomic bootstrapping methods in this vein:

- using the Implicit Trust Anchor database (not an autonomic solution because the URL must be securely distributed),
- engaging a human user to authorize the CA certificate using out-of-band data (not an autonomic solution because the human user is involved),
- using a configured Explicit TA database (not an autonomic solution because the distribution of an explicit TA database is not autonomic),
- and using a Certificate-Less TLS mutual authentication method (not an autonomic solution because the distribution of symmetric key material is not autonomic).

These "touch" methods do not meet the requirements for zero-touch.

There are "call home" technologies where the Pledge first establishes a connection to a well known vendor service using a common client-server authentication model. After mutual authentication appropriate credentials to authenticate the target domain are transfered to the Pledge. This creates serveral problems and limitations:

- the pledge requires realtime connectivity to the vendor service,
- the domain identity is exposed to the vendor service (this is a privacy concern),
- the vendor is responsible for making the authorization decisions (this is a liability concern),

BRSKI addresses these issues by defining extensions to the EST protocol for the automated distribution of vouchers.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined for clarity:

domainID: The domain IDentity is the 160-bit SHA-1 hash of the BIT STRING of the subjectPublicKey of the root certificate for the registrars in the domain. This is consistent with the subject key identifier (Section 4.2.1.2 [RFC5280]).

drop ship: The physical distribution of equipment containing the "factory default" configuration to a final destination. In zero-touch scenarios there is no staging or pre-configuration during drop-ship.

imprint: The process where a device obtains the cryptographic key material to identify and trust future interactions with a network. This term is taken from Konrad Lorenz’s work in biology with new ducklings: during a critical period, the duckling would assume that anything that looks like a mother duck is in fact their mother. An equivalent for a device is to obtain the fingerprint of the network’s root certification authority certificate. A device that imprints on an attacker suffers a similar fate to a duckling that imprints on a hungry wolf. Securely imprinting is a primary focus of this document.[imprinting]. The analogy to Lorenz’s work was first noted in [Stajano99theresurrecting].

enrollment: The process where a device presents key material to a network and acquires a network specific identity. For example when a certificate signing request is presented to a certification authority and a certificate is obtained in response.

Pledge: The prospective device, which has an identity installed by a third-party (e.g., vendor, manufacturer or integrator).

Voucher A signed statement from the MASA service that indicates to a Pledge the cryptographic identity of the Registrar it should trust. There are different types of vouchers depending on how that trust asserted. Multiple voucher types are defined in [I-D.ietf-anima-voucher]

Domain: The set of entities that trust a common key infrastructure trust anchor. This includes the Proxy, Registrar, Domain Certificate Authority, Management components and any existing entity that is already a member of the domain.

Domain CA: The domain Certification Authority (CA) provides certification functionalities to the domain. At a minimum it provides certification functionalities to a Registrar and stores the trust anchor that defines the domain. Optionally, it certifies all elements.

Join Registrar (and Coordinator): A representative of the domain that is configured, perhaps autonomically, to decide whether a new device is allowed to join the domain. The administrator of the domain interfaces with a Join Registrar (and Coordinator) to control this process. Typically a Join Registrar is "inside" its domain. For simplicity this document often refers to this as just "Registrar". The term JRC is used in common with other bootstrap mechanisms.

Join Proxy: A domain entity that helps the pledge join the domain. A Proxy facilitates communication for devices that find themselves in an environment where they are not provided connectivity until after they are validated as members of the domain. The pledge is unaware that they are communicating with a proxy rather than directly with a Registrar.

MASA Service: A third-party Manufacturer Authorized Signing Authority (MASA) service on the global Internet. The MASA signs vouchers. It also provides a repository for audit log information of privacy protected bootstrapping events. It does not track ownership.

Ownership Tracker: An Ownership Tracker service on the global internet. The Ownership Tracker uses business processes to accurately track ownership of all devices shipped against domains that have purchased them. Although optional this component allows vendors to provide additional value in cases where their sales and distribution channels allow for accurately tracking of such ownership. Ownership tracking information is indicated in vouchers as described in [I-D.ietf-anima-voucher]

IDevID: An Initial Device Identity X.509 certificate installed by the vendor on new equipment.

TOFU: Trust on First Use. Used similarly to [RFC7435]. This is where a Pledge device makes no security decisions but rather

simply trusts the first Registrar it is contacted by. This is also known as the "resurrecting duckling" model.

1.3. Scope of solution

Questions have been posed as to whether this solution is suitable in general for Internet of Things (IoT) networks. This depends on the capabilities of the devices in question. The terminology of [RFC7228] is best used to describe the boundaries.

The solution described in this document is aimed in general at non-constrained (i.e. class 2+) devices operating on a non-Challenged network. The entire solution as described here is not intended to be useable as-is by constrained devices operating on challenged networks (such as 802.15.4 LLNs).

In many target applications, the systems involved are large router platforms with multi-gigabit inter-connections, mounted in controlled access data centers. But this solution is not exclusive to the large, it is intended to scale to thousands of devices located in hostile environments, such as ISP provided CPE devices which are drop-shipped to the end user. The situation where an order is fulfilled from distributed warehouse from a common stock and shipped directly to the target location at the request of the domain owner is explicitly supported. That stock ("SKU") could be provided to a number of potential domain owners, and the eventual domain owner will not know a-priori which device will go to which location.

The bootstrapping process can take minutes to complete depending on the network infrastructure and device processing speed. The network communication itself is not optimized for speed; for privacy reasons, the discovery process allows for the Pledge to avoid announcing it’s presence through broadcasting.

This protocol is not intended for low latency handoffs. In networks requiring such things, the pledge SHOULD already have been enrolled.

Specifically, there are protocol aspects described here which might result in congestion collapse or energy-exhaustion of intermediate battery powered routers in an LLN. Those types of networks SHOULD NOT use this solution. These limitations are predominately related to the large credential and key sizes required for device authentication. Defining symmetric key techniques that meet the operational requirements is out-of-scope but the underlying protocol operations (TLS handshake and signing structures) have sufficient algorithm agility to support such techniques when defined.

The imprint protocol described here could, however, be used by non-energy constrained devices joining a non-constrained network (for instance, smart light bulbs are usually mains powered, and speak 802.11). It could also be used by non-constrained devices across a non-energy constrained, but challenged network (such as 802.15.4). The certificate contents, and the process by which the four questions above are resolved do apply to constrained devices. It is simply the actual on-the-wire imprint protocol which could be inappropriate.

This document presumes that network access control has either already occurred, is not required, or is integrated by the proxy and registrar in such a way that the device itself does not need to be aware of the details. Although the use of an X.509 Initial Device Identity is consistant with IEEE 802.1AR [IDevID], and allows for alignment with 802.1X network access control methods, its use here is for Pledge authentication rather than network access control. Integrating this protocol with network access control, perhaps as an Extensible Authentication Protocol (EAP) method (see [RFC3748]), is out-of-scope.

1.4. Leveraging the new key infrastructure / next steps

As a result of the protocol described herein the bootstrapped devices have a common trust anchor and a certificate has optionally been issued from a local PKI. This makes it possible to automatically deploy services across the domain in a secure manner.

Services which benefit from this:

- Device management.
- Routing authentication.
- Service discovery.

The major beneficiary is that it possible to use the credentials deployed by this protocol to secure the Autonomic Control Plane (ACP) ([I-D.ietf-anima-autonomic-control-plane]).

2. Architectural Overview

The logical elements of the bootstrapping framework are described in this section. Figure 1 provides a simplified overview of the components. Each component is logical and may be combined with other components as necessary.

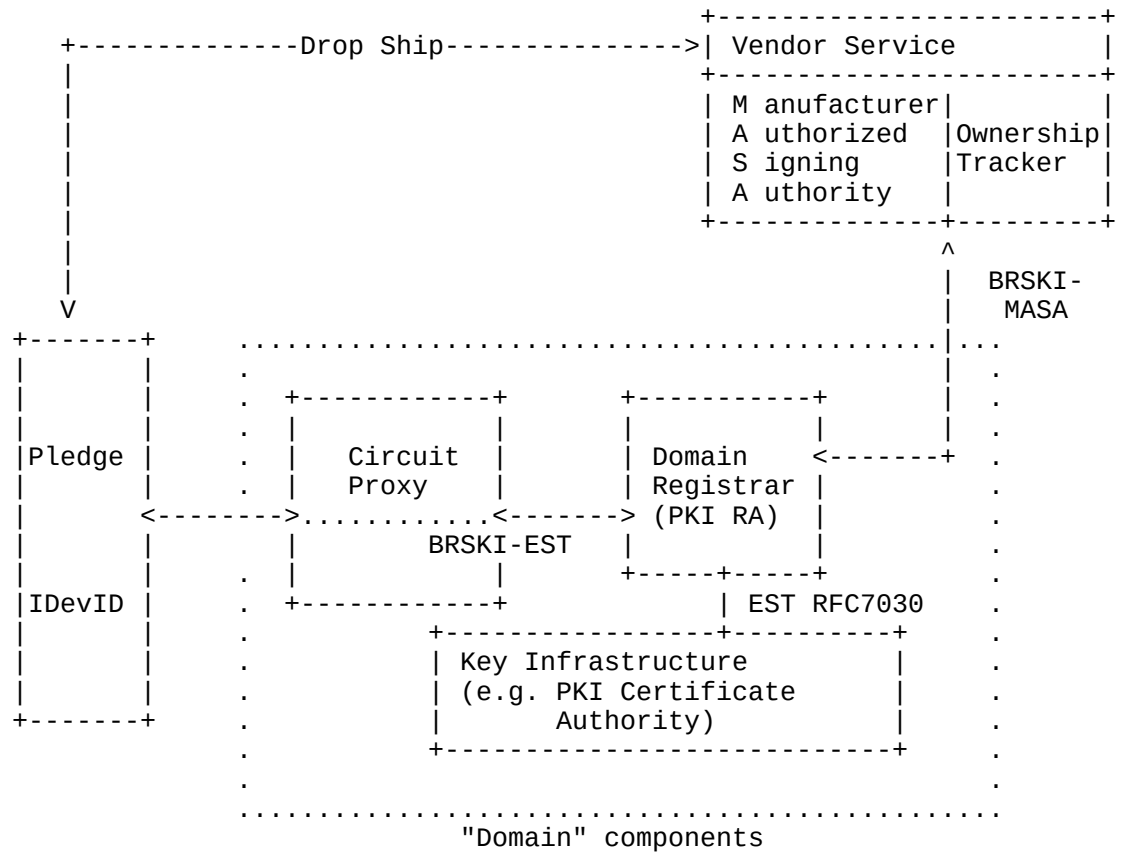


Figure 1

We assume a multi-vendor network. In such an environment there could be a Vendor Service for each vendor that supports devices following this document's specification, or an integrator could provide a generic service authorized by multiple vendors. It is unlikely that an integrator could provide Ownership Tracking services for multiple vendors due to the required sales channel integrations necessary to track ownership.

The domain is the managed network infrastructure with a Key Infrastructure the Pledge is joining. The a domain provides initial device connectivity sufficient for bootstrapping with a Circuit Proxy. The Domain Registrar authenticates the Pledge, makes authorization decisions, and distributes vouchers obtained from the Vendor Service. Optionally the Registrar also acts as a PKI Registration Authority.

2.1. Behavior of a Pledge

The pledge goes through a series of steps which are outlined here at a high level.

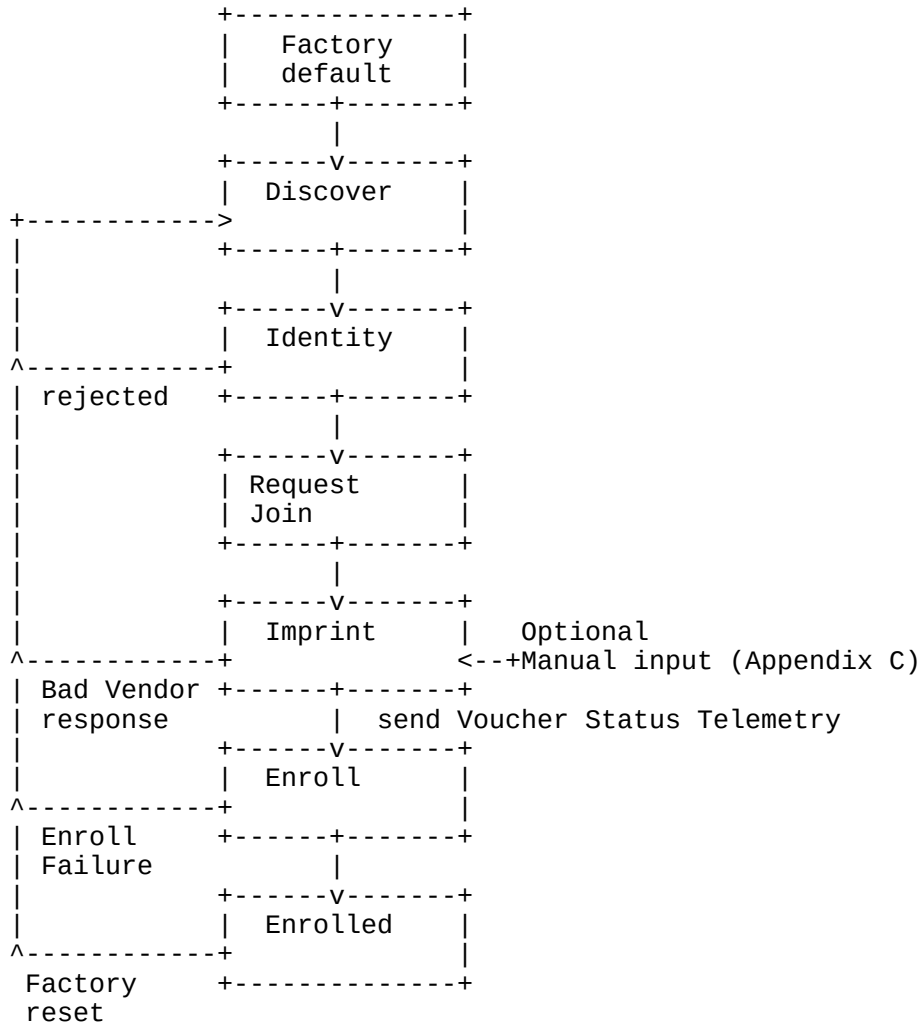


Figure 2

State descriptions for the pledge are as follows:

1. Discover a communication channel to a Registrar.

2. Identify itself. This is done by presenting an X.509 IDevID credential to the discovered Registrar (via the Proxy) in a TLS handshake. (The Registrar credentials are only provisionally accepted at this time).

3. Requests to Join the discovered Registrar. A unique nonce can be included ensuring that any responses can be associated with this particular bootstrapping attempt.

4. Imprint on the Registrar. This requires verification of the vendor service provided voucher. A voucher contains sufficient information for the Pledge to complete authentication of a Registrar. (It enables the Pledge to finish authentication of the Registrar TLS server certificate).

5. Enroll. By accepting the domain specific information from a Registrar, and by obtaining a domain certificate from a Registrar using a standard enrollment protocol, e.g. Enrollment over Secure Transport (EST) [RFC7030].

6. The Pledge is now a member of, and can be managed by, the domain and will only repeat the discovery aspects of bootstrapping if it is returned to factory default settings.

2.2. Secure Imprinting using Vouchers

A voucher is a cryptographically protected statement to the Pledge device authorizing a zero-touch imprint on the Registrar domain.

The format and cryptographic mechanism of vouchers is described in detail in [I-D.ietf-anima-voucher].

Vouchers provide a flexible mechanism to secure imprinting: the Pledge device only imprints when a voucher can be validated. At the lowest security levels the MASA server can indiscriminately issue vouchers. At the highest security levels issuance of vouchers can be integrated with complex sales channel integrations that are beyond the scope of this document. This provides the flexibility for a number of use cases via a single common protocol mechanism on the Pledge and Registrar devices that are to be widely deployed in the field. The MASA vendor services have the flexibility to leverage either the currently defined claim mechanisms or to experiment with higher or lower security levels.

Vouchers provide a signed but non-encrypted communication channel between the Pledge, the MASA, and the Registrar. The Registrar maintains control over the transport and policy decisions allowing the local security policy of the domain network to be enforced.

2.3. Initial Device Identifier

Pledge authentication and Pledge voucher-request signing is via an X.509 certificate installed during the manufacturing process. This Initial Device Identifier provides a basis for authenticating the Pledge during subsequent protocol exchanges and informing the Registrar of the MASA URI. There is no requirement for a common root PKI hierarchy. Each device vendor can generate their own root certificate.

The following previously defined fields are in the X.509 IDevID certificate:

- The subject field's DN encoding MUST include the "serialNumber" attribute with the device's unique serial number.

- The subject-alt field's encoding SHOULD include a non-critical version of the RFC4108 defined HardwareModuleName.

In order to build the voucher "serial-number" field these IDevID fields need to be converted into a serial-number of "type string". The following methods is used depending on the first available IDevID certificate field (attempted in this order):

- An RFC4514 String Representation of the Distinguished Name "serialNumber" attribute.

- The HardwareModuleName hwSerialNum OCTET STRING base64 encoded.

- The RFC4514 String Representation of the Distinguished Name "common name" attribute.

The following newly defined field SHOULD be in the X.509 IDevID certificate: An X.509 non-critical certificate extension that contains a single Uniform Resource Identifier (URI) that points to an on-line Manufacturer Authorized Signing Authority. The URI is represented as described in Section 7.4 of [RFC5280].

Any Internationalized Resource Identifiers (IRIs) MUST be mapped to URIs as specified in Section 3.1 of [RFC3987] before they are placed in the certificate extension. The URI provides the authority information. The BRSKI .well-known tree is described in Section 5

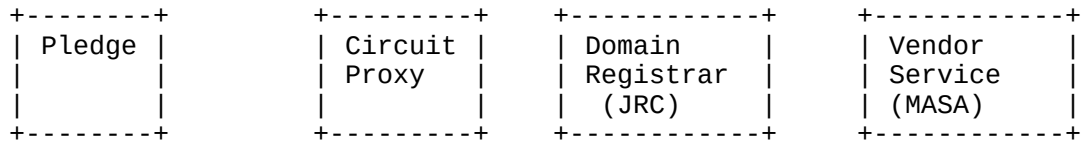
The new extension is identified as follows:

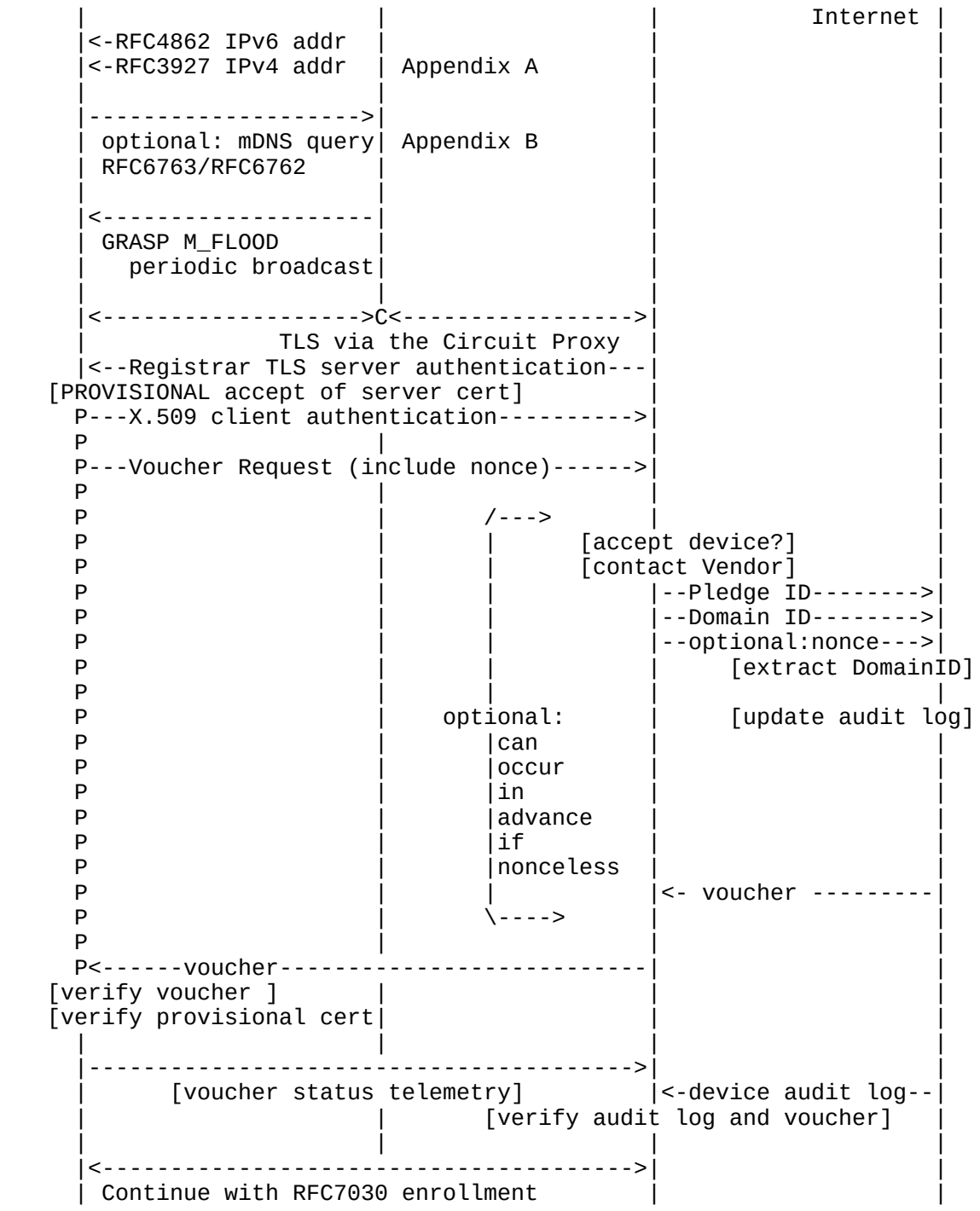
```
<CODE BEGINS>

MASAURLExtnModule-2016 { iso(1) identified-organization(3) dod(6)
internet(1) security(5) mechanisms(5) pkix(7)
id-mod(0) id-mod-MASAURLExtn2016(TBD) }

DEFINITIONS IMPLICIT TAGS ::= BEGIN

-- EXPORTS ALL --

IMPORTS
EXTENSION
FROM PKIX-CommonTypes-2009
{ iso(1) identified-organization(3) dod(6) internet(1)
security(5) mechanisms(5) pkix(7) id-mod(0)
id-mod-pkixCommon-02(57) }

id-pe
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) } ;
MASACertExtensions EXTENSION ::= { ext-MASAURL, ... }
ext-MASAURL EXTENSION ::= { SYNTAX MASAURLSyntax
IDENTIFIED BY id-pe-masa-url }

id-pe-masa-url OBJECT IDENTIFIER ::= { id-pe TBD }

MASAURLSyntax ::= IA5String

END

<CODE ENDS>
```

The choice of id-pe is based on guidance found in Section 4.2.2 of [RFC5280], "These extensions may be used to direct applications to on-line information about the issuer or the subject". The MASA URL is precisely that: online information about the particular subject.

2.4. Protocol Flow

A representative flow is shown in Figure 3:




Internet
<-RFC4862 IPv6 addr
<-RFC3927 IPv4 addr
Appendix A
optional: mDNS query
RFC6763/RFC6762
Appendix B
GRASP M_FLOOD
periodic broadcast
TLS via the Circuit Proxy
<--Registrar TLS server authentication---
[PROVISIONAL accept of server cert]
P---X.509 client authentication---------->
P---Voucher Request (include nonce)------>
[accept device?]
[contact Vendor]
--Pledge ID-------->
--Domain ID-------->
--optional:nonce--->
[extract DomainID]
[update audit log]
optional:
can
occur
in
advance
if
nonceless
<- voucher ---------
P<------voucher----------
[verify voucher]
[verify provisional cert]
[voucher status telemetry]
<-device audit log--
[verify audit log and voucher]
Continue with RFC7030 enrollment

```
   | using now bidirectionally authenticated |                  |
   | TLS session.        |                   |                  |
   |                     |                   |                  |
```

Figure 3

2.4.1. Architectural component: Pledge

The Pledge is the device which is attempting to join. Until the pledge completes the enrollment process, it does has network connectivity only to the Proxy.

2.4.2. Architectural component: Circuit Proxy

The (Circuit) Proxy provides HTTPS connectivity between the pledge and the registrar. The proxy mechanism is described in Section 4, with an optional stateless mechanism described in Appendix C.

2.4.3. Architectural component: Domain Registrar

The Domain Registrar (having the formal name Join Registrar/ Coordinator (JRC)), operates as a CMC Registrar, terminating the EST and BRSKI connections. The Registrar is manually configured or distributed with a list of trust anchors necessary to authenticate any Pledge device expected on the network. The Registrar communicates with the Vendor supplied MASA to establish ownership.

2.4.4. Architectural component: Vendor Service

The Vendor Service provides two logically seperate functions: the Manufacturer Authorized Signing Authority (MASA), and an ownership tracking/auditing function.

2.5. Lack of realtime clock

Many devices when bootstrapping do not have knowledge of the current time. Mechanisms like Network Time Protocols can not be secured until bootstrapping is complete. Therefore bootstrapping is defined in a method that does not require knowledge of the current time.

Unfortunately there are moments during bootstrapping when certificates are verified, such as during the TLS handshake, where validity periods are confirmed. This paradoxical "catch-22" is resolved by the Pledge maintaining a concept of the current "window" of presumed time validity that is continually refined throughout the bootstrapping process as follows:

- Initially the Pledge does not know the current time.

- During Pledge authentiation by the Registrar a realtime clock can be used by the Registrar. This bullet expands on a closely related issue regarding Pledge lifetimes. RFC5280 indicates that long lived Pledge certifiates "SHOULD be assigned the GeneralizedTime value of 99991231235959Z" [RFC7030] so the Registrar MUST support such lifetimes and SHOULD support ignoring Pledge lifetimes if they did not follow the RFC5280 recommendations.

- The Pledge authenticates the voucher presented to it. During this authentication the Pledge ignores certificate lifetimes (by necessity because it does not have a realtime clock).

- If the voucher contains a nonce then the Pledge MUST confirm the nonce matches the original Pledge voucher-request. This ensures the voucher is fresh. See / (Section 5.2).

- Once the voucher is accepted the validity period of the pinned-domain-cert in the voucher now serves as a valid time window. Any subsequent certificate validity periods checked during RFC5280 path validation MUST occur within this window.

- When accepting an enrollment certificate the validity period within the new certificate is assumed to be valid by the Pledge. The Pledge is now willing to use this credential for client authentication.

2.6. Cloud Registrar

The Pledge MAY contact a well known URI of a cloud Registrar if a local Registrar can not be discovered or if the Pledge's target use cases do not include a local Registrar.

If the Pledge uses a well known URI for contacting a cloud Registrar an Implicit Trust Anchor database (see [RFC7030]) MUST be used to authenticate service as described in RFC6125. This is consistent with the human user configuration of an EST server URI in [RFC7030] which also depends on RFC6125.

2.7. Determining the MASA to contact

The registrar needs to be able to contact a MASA that is trusted by the Pledge in order to obtain vouchers. There are three mechanisms described:

The device's Initial Device Identifier will normally contain the MASA URL as detailed in Section 2.3. This is the RECOMMENDED mechanism.

If the Registrar is integrated with [I-D.ietf-opsawg-mud] and the Pledge IDevID contains the id-pe-mud-url then the Registrar MAY attempt to obtain the MASA URL from the MUD file. The MUD file extension for the MASA URL is defined in Appendix D.

It can be operationally difficult to ensure the necessary X.509 extensions are in the Pledge's' IDevID due to the difficulty of aligning current Pledge manufacturing with software releases and development. As a final fallback the Registrar MAY be manually configured or distributed with a MASA URL for each vendor. Note that the Registrar can only select the configured MASA URL based on the trust anchor -- so vendors can only leverage this approach if they ensure a single MASA URL works for all Pledge's associated with each trust anchor.

3. Voucher-Request artifact

The Pledge voucher-request is how a Pledge requests a voucher. The Pledge forms a voucher-request and submits it to the Registrar. The Registrar in turn submits a voucher-request to the MASA server. To help differentiate this document refers to "Pledge voucher-request" and "Registrar voucher-request" when indicating the source is beneficial. The "proximity-registrar-cert" leaf is used in Pledge voucher-requests. The "prior-signed-voucher-request" is used in Registrar voucher-requests that include a Pledge voucher-request.

Unless otherwise signaled (outside the voucher-request artifact), the signing structure is as defined for vouchers, see [I-D.ietf-anima-voucher].

3.1. Tree Diagram

The following tree diagram illustrates a high-level view of a voucher-request document. The notation used in this diagram is described in [I-D.ietf-anima-voucher]. Each node in the diagram is fully described by the YANG module in Section 3.3. Please review the YANG module for a detailed description of the voucher-request format.

```
module: ietf-voucher-request

  grouping voucher-request-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
       +---- prior-signed-voucher-request?   binary
       +---- proximity-registrar-cert?       binary
```

3.2. Examples

This section provides voucher examples for illustration purposes. That these examples conform to the encoding rules defined in [RFC7951].

Example (1) The following example illustrates a Pledge voucher-request. The assertion leaf is indicated as 'proximity' and the Registrar's TLS server certificate is included in the 'proximity-registrar-cert' leaf. See Section 5.2.

```
{
    "ietf-voucher-request:voucher": {
        "nonce": "62a2e7693d82fcda2624de58fb6722e5",
        "created-on": "2017-01-01T00:00:00.000Z",
        "assertion": "proximity",
        "proximity-registrar-cert": "base64encodedvalue=="
    }
}
```

Example (2) The following example illustrates a Registrar voucher-request. The 'prior-signed-voucher-request' leaf is populated with the Pledge's voucher-request (such as the prior example). See Section 5.4.

```
{
    "ietf-voucher-request:voucher": {
        "nonce": "62a2e7693d82fcda2624de58fb6722e5",
        "created-on": "2017-01-01T00:00:02.000Z",
        "assertion": "proximity",
        "idevid-issuer": "base64encodedvalue=="
        "serial-number": "JADA123456789"
        "prior-signed-voucher": "base64encodedvalue=="
    }
}
```

Example (3) The following example illustrates a Registrar voucher-request. The 'prior-signed-voucher-request' leaf is not populated with the Pledge's voucher-request nor is the nonce leaf. This form might be used by a Registrar requesting a voucher when the Pledge is offline or when the Registrar expects to be offline during deployment. See Section 5.4.

```
{
    "ietf-voucher-request:voucher": {
        "created-on": "2017-01-01T00:00:02.000Z",
        "assertion": "TBD",
        "idevid-issuer": "base64encodedvalue=="
        "serial-number": "JADA123456789"
    }
}
```

Example (4) The following example illustrates a Registrar voucher-request. The 'prior-signed-voucher-request' leaf is not populated with the Pledge voucher-request because the Pledge did not sign it's own request. This form might be used when more constrained Pledges are being deployed. The nonce is populated from the Pledge's request. See Section 5.4.

```
{
    "ietf-voucher-request:voucher": {
        "nonce": "62a2e7693d82fcda2624de58fb6722e5",
        "created-on": "2017-01-01T00:00:02.000Z",
        "assertion": "proximity",
        "idevid-issuer": "base64encodedvalue=="
        "serial-number": "JADA123456789"
    }
}
```

3.3. YANG Module

Following is a YANG [RFC7950] module formally extending the [I-D.ietf-anima-voucher] voucher into a voucher-request.

```
<CODE BEGINS> file "ietf-voucher-request@2017-10-30.yang"
module ietf-voucher-request {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-voucher-request";
  prefix "vch";

  import ietf-restconf {
    prefix rc;
    description
      "This import statement is only present to access
       the yang-data extension defined in RFC 8040.";
    reference "RFC 8040: RESTCONF Protocol";
  }

  import ietf-voucher {
    prefix v;
    description
      "FIXME";
    reference "RFC ????: Voucher Profile for Bootstrapping Protocols";
  }

  organization
   "IETF ANIMA Working Group";

  contact
   "WG Web:   <http://tools.ietf.org/wg/anima/>
    WG List:  <mailto:anima@ietf.org>
    Author:   Kent Watsen
              <mailto:kwatsen@juniper.net>
    Author:   Max Pritikin
              <mailto:pritikin@cisco.com>
    Author:   Michael Richardson
              <mailto:mcr+ietf@sandelman.ca>
    Author:   Toerless Eckert
              <mailto:tte+ietf@cs.fau.de>";

  description
   "This module... FIXME

    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.

    Copyright (c) 2017 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or without
    modification, is permitted pursuant to, and subject to the license
    terms contained in, the Simplified BSD License set forth in Section
    4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX; see the RFC
    itself for full legal notices.";

  revision "2017-10-30" {
    description
     "Initial version";
    reference
     "RFC XXXX: Voucher Profile for Bootstrapping Protocols";
  }

  // Top-level statement
  rc:yang-data voucher-request-artifact {
    uses voucher-request-grouping;
  }

  // Grouping defined for future usage
  grouping voucher-request-grouping {
    description
      "Grouping to allow reuse/extensions in future work.";

    uses v:voucher-artifact-grouping {
      refine "voucher/created-on" {
        mandatory false;
      }

      refine "voucher/pinned-domain-cert" {
        mandatory false;
      }

      augment "voucher"  {
        description
          "Adds leaf nodes appropriate for requesting vouchers.";

        leaf prior-signed-voucher-request {
          type binary;
          description
            "If it is necessary to change a voucher, or re-sign and
```

```
              forward a voucher that was previously provided along a
              protocol path, then the previously signed voucher SHOULD be
              included in this field.

              For example, a pledge might sign a proximity voucher, which
              an intermediate registrar then re-signs to make its own
              proximity assertion.  This is a simple mechanism for a
              chain of trusted parties to change a voucher, while
              maintaining the prior signature information.

              The pledge MUST ignore all prior voucher information when
              accepting a voucher for imprinting. Other parties MAY
              examine the prior signed voucher information for the
              purposes of policy decisions. For example this information
              could be useful to a MASA to determine that both pledge and
              registrar agree on proximity assertions. The MASA SHOULD
              remove all prior-signed-voucher information when signing
              a voucher for imprinting so as to minimize the final
              voucher size.";
          }

          leaf proximity-registrar-cert {
            type binary;
            description
              "An X.509 v3 certificate structure as specified by RFC 5280,
               Section 4 encoded using the ASN.1 distinguished encoding
               rules (DER), as specified in ITU-T X.690.

               The first certificate in the Registrar TLS server
               certificate_list sequence  (see [RFC5246]) presented by
               the Registrar to the Pledge. This MUST be populated in a
               Pledge's voucher request if the proximity assertion is
               populated.";
          }
        }
      }
    }
  }

}

<CODE ENDS>
```

4. Proxy details

The role of the Proxy is to facilitate communications. The Proxy forwards packets between the Pledge and a Registrar that has been configured on the Proxy.

The Proxy does not terminate the TLS handshake: it passes streams of bytes onward without examination.

A proxy MAY assume TLS framing for auditing purposes, but MUST NOT assume any TLS version.

A Proxy is always assumed even if it is directly integrated into a Registrar. (In a completely autonomic network, the Registrar MUST provide proxy functionality so that it can be discovered, and the network can grow concentrically around the Registrar)

As a result of the Proxy Discovery process in section Section 4.1.1, the port number exposed by the proxy does not need to be well known, or require an IANA allocation.

If the Proxy joins an Autonomic Control Plane ([I-D.ietf-anima-autonomic-control-plane]) it SHOULD use Autonomic Control Plane secured GRASP ([I-D.ietf-anima-grasp]) to discovery the Registrar address and port. As part of the discovery process, the proxy mechanism (Circuit Proxy vs IPIP encapsulation) is agreed to between the Registrar and Join Proxy.

For the IPIP encapsulation methods (described in Appendix C), the port announced by the Proxy SHOULD be the same as on the registrar in order for the proxy to remain stateless.

In order to permit the proxy functionality to be implemented on the maximum variety of devices the chosen mechanism SHOULD use the minimum amount of state on the proxy device. While many devices in the ANIMA target space will be rather large routers, the proxy function is likely to be implemented in the control plane CPU of such a device, with available capabilities for the proxy function similar to many class 2 IoT devices.

The document [I-D.richardson-anima-state-for-joinrouter] provides a more extensive analysis and background of the alternative proxy methods.

4.1. Pledge discovery of Proxy

The result of discovery is a logical communication with a Registrar, through a Proxy. The Proxy is transparent to the Pledge but is always assumed to exist.

To discover the Proxy the Pledge performs the following actions:

1. MUST: Obtains a local address using IPv6 methods as described in [RFC4862] IPv6 Stateless Address AutoConfiguration. Use of

[RFC4941] temporary addresses is encouraged. A new temporary address SHOULD be allocated whenever the discovery process is forced to restart due to failures. Pledges will generally prefer use of IPv6 Link-Local addresses, and discovery of Proxy will be by Link-Local mechanisms. IPv4 methods are described in Appendix A

2. MUST: Listen for GRASP M_FLOOD ([I-D.ietf-anima-grasp]) announcements of the objective: "AN_Proxy". See section Section 4.1.1 for the details of the objective. The Pledge may listen concurrently for other sources of information, see Appendix B.

Once a proxy is discovered the Pledge communicates with a Registrar through the proxy using the bootstrapping protocol defined in Section 5.

Each discovery method attempted SHOULD exponentially back-off attempts (to a maximum of one hour) to avoid overloading the network infrastructure with discovery. The back-off timer for each method MUST be independent of other methods.

Methods SHOULD be run in parallel to avoid head of queue problems wherein an attacker running a fake proxy or registrar can operate protocol actions intentionally slowly.

Once a connection to a Registrar is established (e.g. establishment of a TLS session key) there are expectations of more timely responses, see Section 5.2.

Once all discovered services are attempted the device SHOULD return to listening for GRASP M_FLOOD. It should periodically retry the vendor specific mechanisms. The Pledge MAY prioritize selection order as appropriate for the anticipated environment.

4.1.1. Proxy Grasp announcements

A proxy uses the GRASP M_FLOOD mechanism to announce itself. The pledge SHOULD listen for messages of these form. This announcement can be within the same message as the ACP announcement detailed in [I-D.ietf-anima-autonomic-control-plane].

```
proxy-objective = ["AN_Proxy", [ O_IPv6_LOCATOR, ipv6-address,
transport-proto, port-number ] ]

ipv6-address       - the v6 LL of the proxy
transport-proto    - 6, for TCP 17 for UDP
port-number        - the TCP or UDP port number to find the proxy
```

Figure 5

4.2. CoAP connection to Registrar

The use of CoAP to connect from Pledge to Registrar is out of scope for this document, and may be described in future work.

4.3. HTTPS proxy connection to Registrar

The proxy SHOULD also provide one of: an IPIP encapsulation of HTTP traffic to the registrar, or a TCP circuit proxy that connects the Pledge to a Registrar.

When the Proxy provides a circuit proxy to a Registrar the Registrar MUST accept HTTPS connections.

4.4. Proxy discovery of Registrar

The Registrar SHOULD announce itself so that proxies can find it and determine what kind of connections can be terminated.

The registrar announces itself using GRASP M_FLOOD messages. The M_FLOOD is formatted as follows:

```
[M_FLOOD, 12340815, h'fda379a6f6ee0000200000064000001', 180000,
            ["AN_join_registrar", 4, 255, "EST-TLS"],
            [O_IPv6_LOCATOR,
                 h'fda379a6f6ee0000200000064000001', TCP, 80
```

Figure 6: Registrar Discovery

The formal CDDL definition is:

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

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

initiator = ACP address to contact Registrar
objective-flags = sync-only  ; as in GRASP spec
sync-only =  4               ; M_FLOOD only requires synchronization
loop-count      = 255        ; mandatory maximum
objective-value = text       ; name of the (list of) of supported
                             ; protocols: "EST-TLS" for RFC7030.
```

Figure 7: AN_join_registrar CDDL

The M_FLOOD message MUST be sent periodically. The period is subject to network administrator policy (EST server configuration). It must be so low that the aggregate amount of periodic M_FLOODs from all EST servers causes negligible traffic across the ACP.

The locators are to be interpreted as follows:

```
locator1  = [O_IPv6_LOCATOR, fd45:1345::6789, 6,  443]
locator2  = [O_IPv6_LOCATOR, fd45:1345::6789, 17, 5683]
locator3  = [O_IPv6_LOCATOR, fe80::1234, 41, nil]
```

Figure 7: Registrar Response

The set of locators is to be interpreted as follows. A protocol of 6 indicates that TCP proxying on the indicated port is desired. A protocol of 17 indicates that UDP proxying on the indicated port is desired. In each case, the traffic SHOULD be proxied to the same port at the ULA address provided.

A protocol of 41 indicates that packets may be IPIP proxy’ed. In the case of that IPIP proxying is used, then the provided link-local address MUST be advertised on the local link using proxy neighbour discovery. The Join Proxy MAY limit forwarded traffic to the protocol (6 and 17) and port numbers indicated by locator1 and locator2. The address to which the IPIP traffic should be sent is the initiator address (an ACP address of the Registrar), not the address given in the locator.

Registrars MUST accept TCP / UDP traffic on the ports given at the ACP address of the Registrar. If the Registrar supports IPIP ntunnelling, it MUST also accept traffic encapsulated with IPIP.

Registrars MUST accept HTTPS/EST traffic on the TCP ports indicated. Registrars MAY accept DTLS/CoAP/EST traffic on the UDP in addition to TCP traffic.

5. Protocol Details

The Pledge MUST initiate BRSKI after boot if it is unconfigured. The Pledge MUST NOT automatically initiate BRSKI if it has been configured or is in the process of being configured.

BRSKI is described as extensions to EST [RFC7030] to reduce the number of TLS connections and crypto operations required on the Pledge. The Registrar implements the BRSKI REST interface within the same .well-known URI tree as the existing EST URIs as described in EST [RFC7030] section 3.2.2. The communication channel between the Pledge and the Registrar is referred to as "BRSKI-EST" (see Figure 1).

The communication channel between the Registrar and MASA is similarly described as extensions to EST within the same ./well-known tree. For clarity this channel is referred to as "BRSKI-MASA". (See Figure 1).

MASA URI is "https:// authority "./well-known/est".

BRSKI uses EST message formats for existing operations, uses JSON [RFC7159] for all new operations defined here, and voucher formats.

While EST section 3.2 does not insist upon use of HTTP 1.1 persistent connections, BRSKI-EST connections SHOULD use persistent connections. The intention of this guidance is to ensure the provisional TLS authentication occurs only once and is properly managed.

Summarized automation extensions for the BRSKI-EST flow are:

- The Pledge provisionally accepts the Registrar certificate during the TLS handshake as detailed in Section 5.1.

- If the Registrar responds with a redirection to other web origins the Pledge MUST follow only a single redirection. (EST supports redirection but does not allow redirections to other web origins without user input).

- The Registar MAY respond with an HTTP 202 ("the request has been accepted for processing, but the processing has not been completed") as described in EST [RFC7030] section 4.2.3 wherein the client "MUST wait at least the specified ’retry-after’ time before repeating the same request". The Pledge is RECOMMENDED to

provide local feed (blinked LED etc) during this wait cycle if mechanisms for this are available. To prevent an attacker Registrar from significantly delaying bootstrapping the Pledge MUST limit the 'retry-after' time to 60 seconds. To avoid blocking on a single erroneous Registrar the Pledge MUST drop the connection after 5 seconds in which there has been no progress on the TCP connection. It should proceed to other discovered Registrars if there are any. If there were no other Registrars discovered, the pledge MAY continue to wait, as long as it is concurrently listening for new proxy announcements.

- Ideally the Pledge could keep track of the appropriate retry-after value for any number of outstanding Registrars but this would involve a large state table on the Pledge. Instead the pledge MAY ignore the exact retry-after value in favor of a single hard coded value that takes effect between discovery ([[ProxyDiscovery]]) attempts. A Registrar that is unable to complete the transaction the first time due to timing reasons will have future chances.

- The Pledge requests and validates a voucher using the new REST calls described below.

- If necessary the Pledge calls the EST defined /cacerts method to obtain the domain owners' CA certificate. The pinned-domain-certificate element from the voucher should validate this certificate, or be identical to it.

- The Pledge completes authentication of the server certificate as detailed in Section 5.5.1. This moves the BRSKI-EST TLS connection out of the provisional state. Optionally, the BRSKI-EST TLS connection can now be used for EST enrollment.

The extensions for a Registrar (equivalent to EST server) are:

- Client authentication is automated using Initial Device Identity (IDevID) as per the EST certificate based client authentication. The subject field's DN encoding MUST include the "serialNumber" attribute with the device's unique serial number. In the language of RFC6125 this provides for a SERIALNUM-ID category of identifier that can be included in a certificate and therefore that can also be used for matching purposes. The SERIALNUM-ID whitelist is collated according to vendor trust anchor since serial numbers are not globally unique.

- The Registrar requests and validates the Voucher from the vendor authorized MASA service.

- The Registrar forwards the Voucher to the Pledge when requested.

- o The Registar performs log verifications in addition to local authorization checks before accepting optional Pledge device enrollment requests.

5.1. BRSKI-EST TLS establishment details

The Pledge establishes the TLS connection with the Registrar through the circuit proxy (see Section 4) but the TLS handshake is with the Registar. The BRSKI-EST Pledge is the TLS client and the BRSKI-EST Registrar is the TLS server. All security associations established are between the Pledge and the Registrar regardless of proxy operations.

Establishment of the BRSKI-EST TLS connection is as specified in EST [RFC7030] section 4.1.1 "Bootstrap Distribution of CA Certificates" [RFC7030] wherein the client is authenticated with the IDevID certificate, and the EST server (the Registrar) is provisionally authenticated with a unverified server certificate.

The Pledge maintains a security paranoia concerning the provisional state, and all data received, until a voucher is received and verified as specified in Section 5.5.1

5.2. Pledge Requests Voucher from the Registrar

When the Pledge bootstraps it makes a request for a Voucher from a Registrar.

This is done with an HTTPS POST using the operation path value of "/.well-known/est/requestvoucher".

The request media types are:

application/pkcs7-mime; smime-type=voucher-request The request is a "YANG-defined JSON document that has been signed using a PKCS#7 structure" as described in Section 3 using the JSON encoding described in [RFC7951]. The Pledge SHOULD sign the request using the Section 2.3 credential.

application/json The request is the "YANG-defined JSON document" as described in Section 3 with exception that it is not within a PKCS#7 structure. It is protected only by the TLS client authentication. This reduces the cryptographic requirements on the Pledge.

For simplicity the term ’voucher-request’ is used to refer to either of these media types. Registrar impementations SHOULD anticipate

future media types but of course will simply fail the request if those types are not yet known.

The Pledge populates the voucher-request fields as follows:

created-on: Pledges that have a realtime clock are RECOMMENDED to populate this field. This provides additional information to the MASA.

nonce: The Pledge voucher-request MUST contain a cryptographically strong random or pseudo-random number nonce. Doing so ensures Section 2.5 functionality. The nonce MUST NOT be reused for bootstrapping attempts.

assertion: The Pledge voucher-request MAY contain an assertion of "proximity".

proximity-registrar-cert: In a Pledge voucher-request this is the first certificate in the TLS server 'certificate_list' sequence (see [RFC5246]) presented by the Registrar to the Pledge. This MUST be populated in a Pledge voucher-request if the "proximity" assertion is populated.

All other fields MAY be omitted in the Pledge voucher-request.

An example JSON payload of a Pledge voucher-request is in Section 3.2 Example 1.

The Registrar validates the client identity as described in EST [RFC7030] section 3.3.2. If the request is signed the Registrar confirms the 'proximity' asserion and associated 'proximity-registrar-cert' are correct. The registrar performs authorization as detailed in [[EDNOTE: UNRESOLVED. See Appendix D "Pledge Authorization"]]. If these validations fail the Registrar SHOULD respond with an appropriate HTTP error code.

If authorization is successful the Registrar obtains a voucher from the MASA service (see Section 5.4) and returns that MASA signed voucher to the pledge as described in Section 5.5.

5.3. BRSKI-MASA TLS establishment details

The BRSKI-MASA TLS connection is a 'normal' TLS connection appropriate for HTTPS REST interfaces. The Registrar initiates the connection and uses the MASA URL obtained as described in Section 2.7 for RFC6125 authentication of the MASA server.

The primary method of Registrar "authentication" by the MASA is detailed in Section 5.4. As detailed in Section 8 the MASA might find it necessary to request additional Registrar authentication. Registrars MUST be prepared to support TLS client certificate authentication and HTTP Basic or Digest authentication as described in RFC7030 for EST clients. Implementors are advised that contacting the MASA is to establish a secured REST connection with a web service and that there are a number of authentication models being explored within the industry. Registrars are RECOMMENDED to fail gracefully and generate useful administrative notifications or logs in the advent of unexpected HTTP 401 (Unauthorized) responses from the MASA.

5.4. Registrar Requests Voucher from MASA

When a Registrar receives a Pledge voucher-request it in turn submits a Registrar voucher-request to the MASA service. For simplicity this is defined as an optional EST message between a Registrar and an EST server running on the MASA service although the Registrar is not required to make use of any other EST functionality when communicating with the MASA service. (The MASA service MUST properly reject any EST functionality requests it does not wish to service; a requirement that holds for any REST interface).

This is done with an HTTP POST using the operation path value of "/.well-known/est/requestvoucher".

The request media type is:

application/pkcs7-mime; smime-type=voucher-request The voucher-request is a "YANG-defined JSON document that has been signed using a PKCS#7 structure" as described in [I-D.ietf-anima-voucher] using the JSON encoding described in [RFC7951]. The Registrar MUST sign the Registrar voucher-request. The entire Registrar certificate chain, up to and including the Domain CA, MUST be included in the PKCS#7 structure.

MASA impementations SHOULD anticipate future media types but of course will simply fail the request if those types are not yet known.

The Registrar populates the voucher-request fields as follows:

created-on: Registrars are RECOMMENDED to populate this field. This provides additional information to the MASA.

nonce: The optional nonce value from the Pledge request if desired (see below).

serial-number: The serial number of the Pledge the Registrar would like a voucher for.

idevid-issuer: The idevid-issuer value from the pledge certificate is included to ensure a statistically unique identity. The Pledge's serial number is extracted from the X.509 IDevID. See Section 2.3.

prior-signed-voucher: If a signed Pledge voucher-request was received then it SHOULD be included in the Registrar voucher-request. (NOTE: what is included is the complete Pledge voucher-request, inclusive of the 'assertion', 'proximity-registrar-cert', etc wrapped by the pledge's original signature).

A nonceless Registrar voucher-request MAY be submitted to the MASA. Doing so allows the Registrar to request a Voucher when the Pledge is offline, or when the Registrar is expected to be offline when the Pledge is being deployed. These use cases require the Registrar to learn the appropriate IDevID SerialNumber field from the physical device labeling or from the sales channel (out-of-scope of this document). If a nonceless voucher-reqeust is submitted the MASA server MUST authenticate the Registrar as described in either EST [RFC7030] section 3.2, section 3.3, or by validating the Registrar's certificate used to sign the Registrar voucher-request. Any of these methods reduce the risk of DDoS attacks and provide an authenticated identity as an input to sales channel integration and authorizations (the actual sale-channel integration is also out-of-scope of this document).

All other fields MAY be omitted in the Registrar voucher-request.

Example JSON payloads of Registrar voucher-requests are in Section 3.2 Example 2 through 4.

The MASA verifies that the Registrar voucher-request is internally consistent but does not necessarily authenticate the Registrar certificate since the registrar is not know to the MASA server in advance. The MASA validation checks before issuing a voucher are as follows:

Renew for expired voucher: As described in [I-D.ietf-anima-voucher] vouchers are normally short lived to avoid revocation issues. If the request is for a previous (expired) voucher using the same Registrar (as determined by the Registrar pinned-domain-cert) and the MASA has not been informed that the claim is invalid then the request for a renewed voucher SHOULD be automatically authorized.

Voucher signature consistency: The MASA MUST verify that the Registrar voucher-request is signed by a Registrar. This is confirmed by verifying that the id-kp-cmcRA extended key usage extension field (as detailed in EST RFC7030 section 3.6.1) exists in the certificate of the entity that signed the Registrar voucher-request. This verification is only a consistency check that the unauthenticated domain CA intended this to be a Registrar. Performing this check provides value to domain PKI by assuring the domain administrator that the MASA service will only respect claims from authorized Registration Authorities of the domain. (The requirement for the Registrar to include the Domain CA certificate in the signature structure was stated above).

Registrar revocation consistency: The MASA SHOULD check for revocation of the Registrar certificate. The maximum lifetime of the voucher issued SHOULD NOT exceed the lifetime of the Registrar's revocation validation (for example if the Registrar revocation status is indicated in a CRL that is valid for two weeks then that is an appropriate lifetime for the voucher). Because the Registar certificate authority is unknown to the MASA in advance this is only an extended consistency check and is not required. The maximum lifetime of the voucher issued SHOULD NOT exceed the lifetime of the Registrar's revocation validation (for example if the Registrar revocation status is indicated in a CRL that is valid for two weeks then that is an appropriate lifetime for the voucher).

Pledge proximity assertion: The MASA server MAY verify that the Registrar voucher-request includes the 'prior-signed-voucher' field populated with a Pledge voucher-request that includes a 'proximity-registrar-cert' that is consistent with the certificate used to sign the Registrar voucher-request. The MASA server is aware of which Pledge's support signing of their voucher requests and can use this information to confirm proximity of the Pledge with the Registrar.

Registar (certificate) authentication: This only occurs if the Registrar voucher-request is nonceless. As noted above the details concerning necessary sales-channel integration for the MASA to authenticate a Registrar certificate is out-of-scope.

The Registrar's certificate chain is extracted from the signature method and the root certificate is used to populate the "pinned-domain-cert" of the Voucher being issued. The domainID (e.g. hash of the root public key) is determined from the pinned-domain-cert and is used to update the audit log.

5.5. Voucher Response

The voucher response to requests from the Pledge and requests from a Registrar are in the same format. A Registrar either caches prior MASA responses or dynamically requests a new Voucher based on local policy.

If the join operation is successful, the server response MUST contain an HTTP 200 response code. The server MUST answer with a suitable 4xx or 5xx HTTP [RFC2616] error code when a problem occurs. The response data from the MASA server MUST be a plaintext human-readable (ASCII, english) error message containing explanatory information describing why the request was rejected.

A 403 (Forbidden) response is appropriate if the voucher-request is not signed correctly, stale, or if the pledge has another outstanding voucher which can not be overridden.

A 404 (Not Found) response is appropriate when the request is for a device which is not known to the MASA.

A 406 (Not Acceptable) response is appropriate if a voucher of the desired type, or using the desired algorithms (as indicated by the Accept: headers, and algorithms used in the signature) can not be issued, such as because the MASA knows the pledge can not process that type.

A 415 (Unsupported Media Type) response is approriate for a request that has a voucher encoding that is not understood.

The response media type is:

application/pkcs7-mime; smime-type=voucher The response is a "YANG-defined JSON document that has been signed using a PKCS#7 structure" as described in [I-D.ietf-anima-voucher] using the JSON encoded described in [RFC7951]. The MASA MUST sign the request.

The syntactic details of vouchers are described in detail in [I-D.ietf-anima-voucher]. For example, the voucher consists of:

```
{
  "ietf-voucher:voucher": {
    "nonce": "62a2e7693d82fcda2624de58fb6722e5",
    "assertion": "logging"
    "pinned-domain-cert": "base64encodedvalue=="
    "serial-number": "JADA123456789"
  }
}
```

The Pledge verifies the signed voucher using the manufacturer installed trust anchor associated with the vendor's selected Manufacturer Authorized Signing Authority.

The 'pinned-domain-cert' element of the voucher contains the domain CA's public key. The Pledge MUST use the 'pinned-domain-cert' trust anchor to immediately complete authentication of the provisional TLS connection.

The Pledge MUST be prepared to parse and fail gracefully from a Voucher response that does not contain a 'pinned-domain-cert' field. The Pledge MUST be prepared to ignore additional fields it does not recognize.

5.5.1. Completing authentication of Provisional TLS connection

If a Registrar's credentials can not be verified using the pinned-domain-cert trust anchor from the voucher then the TLS connection is immediately discarded and the Pledge abandons attempts to bootstrap with this discovered registrar. The pledge SHOULD send voucher status telemetry (described below) before closing the TLS connection. The pledge MUST attempt to enroll using any other proxies it has found. It SHOULD return to the same proxy again after attempting with other proxies. Attempts should be attempted in the exponential backoff described earlier. Attempts SHOULD be repeated as failure may be the result of a temporary inconsistently (an inconsistently rolled Registrar key, or some other mis-configuration). The inconsistently could also be the result an active MITM attack on the EST connection.

The Registrar MUST use a certificate that chains to the pinned-domain-cert as its TLS server certificate.

The Pledge's PKIX path validation of a Registrar certificate's validity period information is as described in Section 2.5. Once the PKIX path validation is successful the TLS connection is no longer provisional.

The pinned-domain-cert is installed as an Explicit Trust Anchor for future operations. It can therefore can be used to authenticate any dynamically discovered EST server that contain the id-kp-cmcRA extended key usage extension as detailed in EST RFC7030 section 3.6.1; but to reduce system complexity the Pledge SHOULD avoid additional discovery operations. Instead the Pledge SHOULD communicate directly with the Registrar as the EST server. The 'pinned-domain-cert' is not a complete distribution of the EST section 4.1.3 CA Certificate Response which is an additional justification for the recommendation to proceed with EST key management operations.

Once a full CA Certificate Response is obtained it is more authoritative for the domain than the limited 'pinned-domain-cert' response.'

5.6. Voucher Status Telemetry

The domain is expected to provide indications to the system administrators concerning device lifecycle status. To facilitate this it needs telemetry information concerning the device's status.

To indicate Pledge status regarding the Voucher, the pledge MUST post a status message.

The posted data media type: application/json

The client HTTP POSTs the following to the server at the EST well known URI /voucher_status. The Status field indicates if the Voucher was acceptable. If it was not acceptable the Reason string indicates why. In the failure case this message is being sent to an unauthenticated, potentially malicious Registrar and therefore the Reason string SHOULD NOT provide information beneficial to an attacker. The operational benefit of this telemetry information is balanced against the operational costs of not recording that an Voucher was ignored by a client the registar expected to continue joining the domain.

```
{
  "version":"1",
  "Status":FALSE /* TRUE=Success, FALSE=Fail"
  "Reason":"Informative human readable message"
  "reason-context": { additional JSON }
}
```

The server SHOULD respond with an HTTP 200 but MAY simply fail with an HTTP 404 error. The client ignores any response. Within the server logs the server SHOULD capture this telemetry information.

The reason-context attribute is an arbitrary JSON object (literal value or hash of values) which provides additional information specific to this pledge. The contents of this field are not subject to standardization."

Additional standard responses MAY be added via Specification Required.

5.7. MASA authorization log Request

After receiving the voucher status telemetry Section 5.6, the Registrar SHOULD request the MASA authorization log from the MASA service using this EST extension. If a device had previously registered with another domain, a Registrar of that domain would show in the log.

This is done with an HTTP GET using the operation path value of "/.well-known/est/requestauditlog".

The Registrar MUST HTTP POSTs the same Registrar voucher-request as it did when requesting a Voucher. It is posted to the /requestauditlog URI instead. The "idevid-issuer" and "serial-number" informs the MASA server which log is requested so the appropriate log can be prepared for the response. Using the same media type and message minimizes cryptographic and message operations although it results in additional network traffic. The relying MASA server implementation MAY leverage internal state to associate this request with the original, and by now already validated, Registrar voucher-request so as to avoid an extra crypto validation.

A MASA which receives a request for a device which does not exist, or for which the requesting owner was never an owner returns an HTTP 404 ("Not found") code.

Rather than returning the audit log as a response to the POST (with a return code 200), the MASA MAY instead return a 201 ("Created") RESTful response ([RFC7231] section 7.1) containing a URL to the prepared (and easily cachable) audit response.

MASA servers that return URLs SHOULD take care to make the returned URL unguessable. URLs containing a database number such as https://example.com/auditlog/1234 or the EUI of the device such https://example.com/auditlog/10-00-00-11-22-33, would be easily enumerable by an attacker. It is recommended put to put some meaningless randomly generated slug that indexes a database instead.

A MASA that returns a code 200 MAY also include a Location: header for future reference by the Registrar.

The request media type is:

application/pkcs7-mime; smime-type=voucher-request The request is a "YANG-defined JSON document that has been signed using a PKCS#7 structure" as described in Section 3 using the JSON encoded described in [RFC7951]. The Registrar MUST sign the request. The

entire Registrar certificate chain, up to and including the Domain CA, MUST be included in the PKCS#7 structure.

5.7.1. MASA authorization log Response

A log data file is returned consisting of all log entries. For example:

```
{
  "version":"1",
  "events":[
    {
     "date":"<date/time of the entry>",
     "domainID":"<domainID extracted from voucher-request>",
     "nonce":"<any nonce if supplied (or the exact string 'NULL')>"
    },
    {
     "date":"<date/time of the entry>",
     "domainID":"<domainID extracted from voucher-request>",
     "nonce":"<any nonce if supplied (or the exact string 'NULL')>"
    }
  ]
}
```

Distribution of a large log is less than ideal. This structure can be optimized as follows: All nonceless entries for the same domainID MAY be condensed into the single most recent nonceless entry.

A Registrar SHOULD use this log information to make an informed decision regarding the continued bootstrapping of the Pledge. For example if the log includes an unexpected domainID then the Pledge could have imprinted on an unexpected domain. If the log includes nonceless entries then any registrar in the same domain could theoretically trigger a reset of the device and take over management of the Pledge. Equipment that is purchased pre-owned can be expected to have an extensive history. A Registrar MAY request logs at future times. A Registrar MAY be configured to ignore the history of the device but it is RECOMMENDED that this only be configured if hardware assisted NEA [RFC5209] is supported.

Log entries can be compared against local history logs in search of discrepancies.

This document specifies a simple log format as provided by the MASA service to the registar. This format could be improved by distributed consensus technologies that integrate vouchers with a technologies such as block-chain or hash trees or the like. Doing so is out of the scope of this document but are anticipated improvements

for future work. As such, the Registrar client SHOULD anticipate new kinds of responses, and SHOULD provide operator controls to indicate how to process unknown responses.

5.8. EST Integration for PKI bootstrapping

The Pledge SHOULD follow the BRSKI operations with EST enrollment operations including "CA Certificates Request", "CSR Attributes" and "Client Certificate Request" or "Server-Side Key Generation" etc. This is a relatively seamless integration since BRSKI REST calls provide an automated alternative to the manual bootstrapping method described in [RFC7030]. As noted above, use of HTTP 1.1 persistent connections simplifies the Pledge state machine.

The Pledge is also RECOMMENDED to implement the following EST automation extensions. They supplement the RFC7030 EST to better support automated devices that do not have an end user.

Although EST allows clients to obtain multiple certificates by sending multiple CSR requests BRSKI mandates use of the CSR Attributes request and mandates that the Registrar validate the CSR against the expected attributes. This implies that client requests will "look the same" and therefore result in a single logical certificate being issued even if the client were to make multiple requests. Registrars MAY contain more complex logic but doing so is out-of-scope of this specification. BRSKI does not signal any enhancement or restriction to this capability. Pledges that require multiple certificates could establish direct EST connections to the Registrar.

5.8.1. EST Distribution of CA Certificates

The Pledge MUST request the full EST Distribution of CA Certificates message. See RFC7030, section 4.1.

This ensures that the Pledge has the complete set of current CA certificates beyond the pinned-domain-cert (see Section 5.5.1 for a discussion of the limitations inherent in having a single certificate instead of a full CA Certificates response). Although these limitations are acceptable during initial bootstrapping they are not appropriate for ongoing PKIX end entity certificate validation.

5.8.2. EST CSR Attributes

Automated bootstrapping occurs without local administrative configuration of the Pledge. In some deployments its plausible that the Pledge generates a certificate request containing only identity information known to the Pledge (essentially the X.509 IDevID

information) and ultimately receives a certificate containing domain specific identity information. Conceptually the CA has complete control over all fields issued in the end entity certificate. Realistically this is operationally difficult with the current status of PKI certificate authority deployments where the CSR is submitted to the CA via a number of non-standard protocols. Even with all standardized protocols used, it could operationally be problematic to expect that service specific certificate fields can be created by a CA that is likely operated by a group that has no insight into different network services/protocols used. For example, the CA could even be outsourced.

To alleviate these operational difficulties, the Pledge MUST request the EST "CSR Attributes" from the EST server and the EST server needs to be able to reply with the attributes necessary for use of the certificate in its intended protocols/services. This approach allows for minimal CA integrations and instead the local infrastructure (EST server) informs the Pledge of the proper fields to include in the generated CSR. This approach is beneficial to automated boostrapping in the widest number of environments.

If the hardwareModuleName in the X.509 IDevID is populated then it SHOULD by default be propagated to the LDevID along with the hwSerialNum. The EST server SHOULD support local policy concerning this functionality.

In networks using the BRSKI enrolled certificate to authenticate the ACP (Autonomic Control Plane), the EST attributes MUST include the "ACP information" field. See [I-D.ietf-anima-autonomic-control-plane] for more details.

The Registar MUST also confirm the resulting CSR is formatted as indicated before forwarding the request to a CA. If the Registar is communicating with the CA using a protocol like full CMC which provides mechanisms to override the CSR attributes, then these mechanisms MAY be used even if the client ignores CSR Attribute guidance.

5.8.3. EST Client Certificate Request

The Pledge MUST request a new client certificate. See RFC7030, section 4.2.

5.8.4. Enrollment Status Telemetry

For automated bootstrapping of devices the adminstrative elements providing bootstrapping also provide indications to the system administrators concerning device lifecycle status. This might

include information concerning attempted bootstrapping messages seen by the client, MASA provides logs and status of credential enrollment. The EST protocol assumes an end user and therefore does not include a final success indication back to the server. This is insufficient for automated use cases.

To indicate successful enrollment the client SHOULD re-negotiate the EST TLS session using the newly obtained credentials. This occurs by the client initiating a new TLS ClientHello message on the existing TLS connection. The client MAY simply close the old TLS session and start a new one. The server MUST support either model.

In the case of a FAIL the Reason string indicates why the most recent enrollment failed. The SubjectKeyIdentifier field MUST be included if the enrollment attempt was for a keypair that is locally known to the client. If EST /serverkeygen was used and failed then the field is omitted from the status telemetry.

In the case of a SUCCESS the Reason string is omitted. The SubjectKeyIdentifier is included so that the server can record the successful certificate distribution.

Status media type: application/json

The client HTTP POSTs the following to the server at the new EST well known URI /enrollstatus.

```
{
  "version":"1",
  "Status":TRUE /* TRUE=Success, FALSE=Fail"
  "Reason":"Informative human readable message"
  "reason-context": "Additional information"
}
```

The server SHOULD respond with an HTTP 200 but MAY simply fail with an HTTP 404 error.

Within the server logs the server MUST capture if this message was received over an TLS session with a matching client certificate. This allows for clients that wish to minimize their crypto operations to simply POST this response without renegotiating the TLS session - at the cost of the server not being able to accurately verify that enrollment was truly successful.

5.8.5. EST over CoAP

This document describes extensions to EST for the purposes of bootstrapping of remote key infrastructures. Bootstrapping is relevant for CoAP enrollment discussions as well. The defintion of EST and BRSKI over CoAP is not discussed within this document beyond ensuring proxy support for CoAP operations. Instead it is anticipated that a definition of CoAP mappings will occur in subsequent documents such as [I-D.vanderstok-ace-coap-est] and that CoAP mappings for BRSKI will be discussed either there or in future work.

6. Reduced security operational modes

A common requirement of bootstrapping is to support less secure operational modes for support specific use cases. The following sections detail specific ways that the Pledge, Registrar and MASA can be configured to run in a less secure mode for the indicated reasons.

6.1. Trust Model

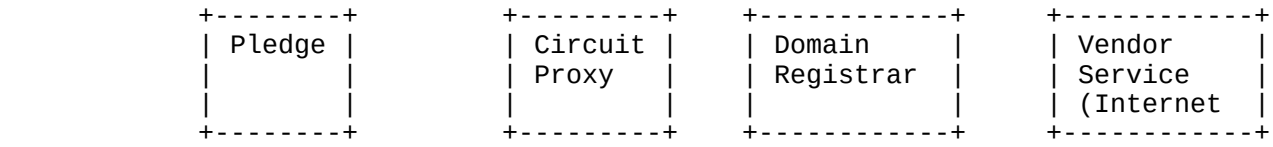


Figure 10

Pledge: The Pledge could be compromised and providing an attack vector for malware. The entity is trusted to only imprint using secure methods described in this document. Additional endpoint assessment techniques are RECOMMENDED but are out-of-scope of this document.

Proxy: Provides proxy functionalities but is not involved in security considerations.

Registrar: When interacting with a MASA server a Registrar makes all decisions. When Ownership Vouchers are involved a Registrar is only a conduit and all security decisions are made on the vendor service.

Vendor Service, MASA: This form of vendor service is trusted to accurately log all claim attempts and to provide authoritative log information to Registrars. The MASA does not know which devices are associated with which domains. These claims could be strengthened by using cryptographic log techniques to provide

append only, cryptographic assured, publicly auditable logs. Current text provides only for a trusted vendor.

Vendor Service, Ownership Validation: This form of vendor service is trusted to accurately know which device is owned by which domain.

6.2. Pledge security reductions

The Pledge can choose to accept vouchers using less secure methods. These methods enable offline and emergency (touch based) deployment use cases:

1. The Pledge MUST accept nonceless vouchers. This allows for offline use cases. Logging and validity periods address the inherent security considerations of supporting these use cases.

2. The Pledge MAY support "trust on first use" for physical interfaces such as a local console port or physical user interface but MUST NOT support "trust on first use" on network interfaces. This is because "trust on first use" permanently degrades the security for all use cases.

3. The Pledge MAY have an operational mode where it skips Voucher validation one time. For example if a physical button is depressed during the bootstrapping operation. This can be useful if the vendor service is unavailable. This behavior SHOULD be available via local configuration or physical presence methods to ensure new entities can always be deployed even when autonomic methods fail. This allows for unsecured imprint.

It is RECOMMENDED that "trust on first use" or skipping voucher validation only be available if hardware assisted Network Endpoint Assessment [RFC5209] is supported. This recommendation ensures that domain network monitoring can detect innappropriate use of offline or emergency deployment procedures.

6.3. Registrar security reductions

A Registrar can choose to accept devices using less secure methods. These methods are acceptable when low security models are needed, as the security decisions are being made by the local administrator, but they MUST NOT be the default behavior:

1. A registrar MAY choose to accept all devices, or all devices of a particular type, at the administrator’s discretion. This could occur when informing all Registrars of unique identifiers of new entities might be operationally difficult.

2. A registrar MAY choose to accept devices that claim a unique identity without the benefit of authenticating that claimed identity. This could occur when the Pledge does not include an X.509 IDevID factory installed credential. New Entities without an X.509 IDevID credential MAY form the Section 5.2 request using the Section 5.4 format to ensure the Pledge's serial number information is provided to the Registar (this includes the IDevID AuthorityKeyIdentifier value which would be statically configured on the Pledge). The Pledge MAY refuse to provide a TLS client certificate (as one is not available). The Pledge SHOULD support HTTP-based or certificate-less TLS authentication as described in EST RFC7030 section 3.3.2. A Registrar MUST NOT accept unauthenticated New Entities unless it has been configured to do so by an administrator that has verified that only expected new entities can communicate with a Registrar (presumably via a physically secured perimeter).

3. A Registrar MAY submit a nonceless voucher-requests to MASA service (by not including a nonce in the voucher-request). The resulting Vouchers can then be stored by the Registrar until they are needed during bootstrapping operations. This is for use cases where target network is protected by an air gap and therefore can not contact the MASA service during Pledge deployment.

4. A registrar MAY ignore unrecognized nonceless log entries. This could occur when used equipment is purchased with a valid history being deployed in air gap networks that required permanent Vouchers.

6.4. MASA security reductions

Lower security modes chosen by the MASA service effect all device deployments unless bound to the specific device identities. In which case these modes can be provided as additional features for specific customers. The MASA service can choose to run in less secure modes by:

1. Not enforcing that a nonce is in the Voucher. This results in distribution of Voucher that never expires and in effect makes the Domain an always trusted entity to the Pledge during any subsequent bootstrapping attempts. That this occurred is captured in the log information so that the Registrar can make appropriate security decisions when a Pledge joins the Domain. This is useful to support use cases where Registrars might not be online during actual device deployment. Because this results in long lived Voucher and does not require the proof that the device is online this is only accepted when the Registrar is

authenticated by the MASA server and authorized to provide this functionality. The MASA server is RECOMMENDED to use this functionality only in concert with an enhanced level of ownership tracking (out-of-scope). If the Pledge device is known to have a real-time-clock that is set from the factory use of a voucher validity period is RECOMMENDED.

2. Not verifying ownership before responding with an Voucher. This is expected to be a common operational model because doing so relieves the vendor providing MASA services from having to track ownership during shipping and supply chain and allows for a very low overhead MASA service. A Registrar uses the audit log information as a defense in depth strategy to ensure that this does not occur unexpectedly (for example when purchasing new equipment the Registrar would throw an error if any audit log information is reported). The MASA should verify the 'prior-signed-voucher' information for Pledge's that support that functionality. This provides a proof-of-proximity check that reduces the need for ownership verification.

7. IANA Considerations

This document requests the following Parameter Values for the "smime-type" Parameters:

- voucher-request
- voucher

7.1. PKIX Registry

IANA is requested to register the following:

This document requests a number for id-mod-MASAURLExtn2016(TBD) from the pkix(7) id-mod(0) Registry. [[EDNOTE: fix names]]

This document requests a number from the id-pe registry for id-pe-masa-url. XXX

7.2. Voucher Status Telemetry

IANA is requested to create a registry entitled: _Voucher Status Telemetry Attributes_. New items can be added using the Specification Required. The following items are to be in the initial registration, with this document as the reference:

- version

- o Status
- o Reason
- o reason-context

8. Security Considerations

There are uses cases where the MASA could be unavailable or uncooperative to the Registrar. They include planned and unplanned network partitions, changes to MASA policy, or other instances where MASA policy rejects a claim. These introduce an operational risk to the Registrar owner that MASA/vendor behavior might limit the ability to re-boostrap a Pledge device. For example this might be an issue during disaster recovery. This risk can be mitigated by Registrars that request and maintain long term copies of "nonceless" Vouchers. In that way they are guaranteed to be able to repeat bootstrapping for their devices.

The issuance of nonceless vouchers themselves create a security concern. If the Registrar of a previous domain can intercept protocol communications then it can use a previously issued nonceless voucher to establish management control of a pledge device even after having sold it. This risk is mitigated by recording the issuance of such vouchers in the MASA audit log that is verified by the subsequent Registrar. This reduces the resale value of the equipment because future owners will detect the lowered security inherent in the existence of a nonceless voucher that would be trusted by their Pledge. This reflects a balance between partition resistant recovery and security of future bootstrapping. Registrars take the Pledge's audit history into account when applying policy to new devices.

The MASA server is exposed to DoS attacks wherein attackers claim an unbounded number of devices. Ensuring a Registrar is representative of a valid vendor customer, even without validating ownership of specific Pledge devices, helps to mitigate this. Pledge signatures on the Pledge voucher-request, as forwarded by the Registrar in the prior-signed-voucher field of the Registrar voucher-request, significantly reduce this risk by ensuring the MASA can confirm proximity between the Pledge and the Registrar making the request. This mechanism is optional to allow for constrained devices.

To facilitate logging and administrative oversight in addition to triggering Registration verification of MASA logs the Pledge reports on Voucher parsing status to the Registrar. In the case of a failure this information is informative to a potentially malicious Registar but this is mandated anyway because of the operational benefits of an informed administrator in cases where the failure is indicative of a

problem. The Registrar is RECOMMENDED to verify MASA logs if voucher status telemetry is not received.

The MASA authorization log includes a hash of the domainID for each registrar a voucher has been issued to. This information is closely related to the actual domain identity, especially when paired with the anti-DDoS authentication information the MASA might collect. This could provide sufficient information for the MASA service to build a detailed understanding the devices that have been provisioned within a domain. There are a number of design choices that mitigate this risk. The domain can maintain some privacy since it has not necessarily been authenticated and is not authoritatively bound to the supply chain. Additionally the domainID captures only the unauthenticated subject key identifier of the domain. A privacy sensitive domain could theoretically generate a new domainID for each device being deployed. Similarly a privacy sensitive domain would likely purchase devices that support proximity assertions from a vendor that does not require sales channel integrations. This would result in a significant level of privacy while maintaining the security characteristics provided by Registrar based audit log inspection.

To facilitate truely limited clients EST RFC7030 section 3.3.2 requirements that the client MUST support a client authentication model have been reduced in Section 6 to a statement that the Registrar "MAY" choose to accept devices that fail cryptographic authentication. This reflects current (poor) practices in shipping devices without a cryptographic identity that are NOT RECOMMENDED.

During the provisional period of the connection the Pledge MUST treat all HTTP header and content data as untrusted data. HTTP libraries are regularly exposed to non-secured HTTP traffic: mature libraries should not have any problems.

Pledge’s might chose to engage in protocol operations with multiple discovered Registrars in parallel. As noted above they will only do so with distinct nonce values, but the end result could be multiple voucher’s issued from the MASA if all registrars attempt to claim the device. This is not a failure and the Pledge choses whichever voucher to accept based on internal logic. The Registrar’s verifying log information will see multiple entries and take this into account for their analytics purposes.

8.1. Freshness in Voucher-Requests

A concern has been raised that the Pledge voucher-request should contain some content (a nonce) provided by the Registrar and/or MASA

in order for those actors to verify that the Pledge voucher-request is fresh.

There are a number of operational problems with getting a nonce from the MASA to the pledge. It is somewhat easier to collect a random value from the Registrar, but as the Registrar is not yet vouched for, such a Registrar nonce has little value. There are privacy and logistical challenges to addressing these operational issues, so if such a thing were to be considered, it would have to provide some clear value. This section examines the impacts of not having a fresh Pledge voucher-request.

Because the Registrar authenticates the Pledge a full Man-in-the-Middle attack is not possible, despite the provisional TLS authentication by the Pledge (see Section 5). Instead we examine the case of a fake Registrar (Rm) that communicates with the Pledge in parallel or in close time proximity with the intended Registrar. (This scenario is intentionally supported as described in Section 4.1).

The fake Registrar (Rm) can obtain a voucher signed by the MASA either directly or through arbitrary intermediaries. Assuming that the MASA accepts the Registar voucher-request (either because Rm is collaborating with a legitimate Registrar according to supply chain information, or because the MASA is in audit-log only mode), then a voucher linking the pledge to the Registrar Rm is issued.

Such a voucher, when passed back to the Pledge, would link the pledge to Registrar Rm, and would permit the Pledge to end the provisional state. It now trusts Rm and, if it has any security vulnerabilities leveragable by an Rm with full administrative control, can be assumed to be a threat against the intended Registrar.

This flow is mitigated by the intended Registar verifying the audit logs available from the MASA as described in Section 5.7. Rm might chose to wait until after the intended Registrar completes the authorization process before submitting the now-stale Pledge voucher-request. The Rm would need to remove the Pledge’s nonce.

In order to successfully use the resulting "stale voucher" Rm would have to attack the Pledge and return it to a bootstrapping enabled state. This would require wiping the Pledge of current configuration and triggering a re-bootstrapping of the Pledge. This is no more likely than simply taking control of the Pledge directly but if this is a consideration the target network is RECOMMENDED to take the following steps:

- o Ongoing network monitoring for unexpected bootstrapping attempts by Pledges.

- o Retreival and examination of MASA log information upon the occurance of any such unexpected events. Rm will be listed in the logs.

9. Acknowledgements

We would like to thank the various reviewers for their input, in particular Brian Carpenter, Toerless Eckert, Fuyu Eleven, Eliot Lear, Sergey Kasatkin, Markus Stenberg, and Peter van der Stok

10. References

10.1. Normative References

[I-D.ietf-anima-autonomic-control-plane]
Behringer, M., Eckert, T., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-ietf-anima-autonomic-control-plane-12 (work in progress), October 2017.

[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.

[I-D.ietf-anima-voucher]
Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "Voucher Profile for Bootstrapping Protocols", draft-ietf-anima-voucher-06 (work in progress), October 2017.

[IDevID] IEEE Standard, "IEEE 802.1AR Secure Device Identifier", December 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>.

[RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, DOI 10.17487/RFC3542, May 2003, <https://www.rfc-editor.org/info/rfc3542>.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <https://www.rfc-editor.org/info/rfc3748>.

[RFC3927] Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses", RFC 3927, DOI 10.17487/RFC3927, May 2005, <https://www.rfc-editor.org/info/rfc3927>.

[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>.

[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>.

[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>.

[RFC5386] Williams, N. and M. Richardson, "Better-Than-Nothing Security: An Unauthenticated Mode of IPsec", RFC 5386, DOI 10.17487/RFC5386, November 2008, <https://www.rfc-editor.org/info/rfc5386>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5660] Williams, N., "IPsec Channels: Connection Latching", RFC 5660, DOI 10.17487/RFC5660, October 2009, <https://www.rfc-editor.org/info/rfc5660>.

[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>.

[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>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[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>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

10.2. Informative References

[I-D.behringer-homenet-trust-bootstrap] Behringer, M., Pritikin, M., and S. Bjarnason, "Bootstrapping Trust on a Homenet", draft-behringer-homenet-trust-bootstrap-02 (work in progress), February 2014.

[I-D.ietf-netconf-zerotouch] Watsen, K., Abrahamsson, M., and I. Farrer, "Zero Touch Provisioning for NETCONF or RESTCONF based Management", draft-ietf-netconf-zerotouch-19 (work in progress), October 2017.

[I-D.ietf-opsawg-mud] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", draft-ietf-opsawg-mud-13 (work in progress), October 2017.

[I-D.richardson-anima-state-for-joinrouter]
Richardson, M., "Considerations for stateful vs stateless join router in ANIMA bootstrap", draft-richardson-anima-state-for-joinrouter-01 (work in progress), July 2016.

[I-D.vanderstok-ace-coap-est]
Kumar, S., Stok, P., Kampanakis, P., Furuhed, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-vanderstok-ace-coap-est-02 (work in progress), June 2017.

[imprinting]
Wikipedia, "Wikipedia article: Imprinting", July 2015, <https://en.wikipedia.org/wiki/Imprinting_(psychology)>.

[RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, DOI 10.17487/RFC2473, December 1998, <https://www.rfc-editor.org/info/rfc2473>.

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

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[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>.

[Stajano99theresurrecting]
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>.

Appendix A. IPv4 operations

A.1. IPv4 Link Local addresses

Instead of an IPv6 link-local address, an IPv4 address may be generated using [RFC3927] Dynamic Configuration of IPv4 Link-Local Addresses.

In the case that an IPv4 Local-Local address is formed, then the bootstrap process would continue as in the IPv6 case by looking for a (circuit) proxy.

A.2. Use of DHCPv4

The Plege MAY obtain an IP address via DHCP [RFC2131]. The DHCP provided parameters for the Domain Name System can be used to perform DNS operations if all local discovery attempts fail.

Appendix B. mDNS / DNSSD proxy discovery options

The Pledge MAY perform DNS-based Service Discovery [RFC6763] over Multicast DNS [RFC6762] searching for the service "_bootstrapks._tcp.local.".

To prevent unaccceptable levels of network traffic the congestion avoidance mechanisms specified in [RFC6762] section 7 MUST be followed. The Pledge SHOULD listen for an unsolicited broadcast response as described in [RFC6762]. This allows devices to avoid announcing their presence via mDNS broadcasts and instead silently join a network by watching for periodic unsolicited broadcast responses.

Performs DNS-based Service Discovery [RFC6763] over normal DNS operations. The service searched for is "_bootstrapks._tcp.example.com". In this case the domain "example.com" is discovered as described in [RFC6763] section 11. This method is only available if the host has received a useable IPv4 address via DHCPv4 as suggested in Appendix A.

If no local bootstrapks service is located using the GRASP mechanisms, or the above mentioned DNS-based Service Discovery methods the Pledge MAY contact a well known vendor provided bootstrapping server by performing a DNS lookup using a well known URI such as "bootstrapks.vendor-example.com". The details of the URI are vendor specific. Vendors that leverage this method on the Pledge are responsible for providing the bootstrapks service.

The current DNS services returned during each query is maintained until bootstrapping is completed. If bootstrapping fails and the Pledge returns to the Discovery state it picks up where it left off and continues attempting bootstrapping. For example if the first Multicast DNS _bootstrapks._tcp.local response doesn't work then the second and third responses are tried. If these fail the Pledge moves on to normal DNS-based Service Discovery.

Appendix C. IPIP Join Proxy mechanism

The Circuit Proxy mechanism suffers from requiring a state on the Join Proxy for each connection that is relayed. The Circuit Proxy can be considered a kind of Algorithm Gateway [FIND-good-REF].

An alternative to proxying at the TCP layer is to selectively forward at the IP layer. This moves all per-connection to the Join Registrar. The IPIP tunnel statelessly forwards packets. This section provides some explanation of some of the details of the Registrar discovery procotol which are not important to Circuit Proxy, and some implementation advice.

The IPIP tunnel is described in [RFC2473]. Each such tunnel is considered a unidirectional construct, but two tunnels may be associated to form a bidirectional mechanism. An IPIP tunnel is setup as follows. The outer addresses are an ACP address of the Join Proxy, and the ACP address of the Join Registrar. The inner addresses seen in the tunnel are the link-local addresses of the network on which the join activity is occuring.

One way to look at this construct is to consider that the Registrar is extending attaching an interface to the network on which the Join Proxy is physically present. The Registrar then interacts as if it were present on that network using link-local (fe80::) addresses. The Join node is unaware that the traffic is being proxied through a tunnel, and does not need any special routing.

There are a number of considerations with this mechanism which require cause some minor amounts of complexity. Note that due to the tunnels, the Registrar sees multiple connections to a fe80::/10 network on not just physical interfaces, but on each of the virtual interfaces represending the tunnels.

C.1. Multiple Join networks on the Join Proxy side

The Join Proxy will in the general case be a routing device with multiple interfaces. Even a device as simple as a wifi access point may have wired, and multiple frequencies of wireless interfaces, potentially with multiple ESSIDs.

Each of these interfaces on the Join Proxy may be seperate L3 routing domains, and therefore will have a unique set of link-local addresses. An IPIP packet being returned by the Registrar needs to be forwarded to the correct interface, so the Join Proxy needs an additional key to distinguish which network the packet should be returned to.

The simplest way to get this additional key is to allocate an additional ACP address; one address for each network on which join traffic is occuring. The Join Proxy SHOULD do a GRASP M_NEG_SYN for each interface which they wish to relay traffic, as this allows the Registrar to do any static tunnel configuration that may be required.

C.2. Automatic configuration of tunnels on Registrar

The Join Proxy is expected to do a GRASP negotiation with the proxy for each Join Interface that it needs to relay traffic from. This is to permit Registrars to configure the appropriate virtual interfaces before join traffic arrives.

A Registrar serving a large number of interfaces may not wish to allocate resources to every interface at all times, but can instead dynamically allocate interfaces. It can do this by monitoring IPIP traffic that arrives on it’s ACP interface, and when packets arrive from new Join Proxys, it can dynamically configure virtual interfaces.

A more sophisticated Registrar willing to modify the behaviour of it’s TCP and UDP stack could note the IPIP traffic origination in the socket control block and make information available to the TCP layer (for HTTPS connections), or to the application (for CoAP connections) via a proprietary extension to the socket API.

C.3. Proxy Neighbor Discovery by Join Proxy

The Join Proxy MUST answer neighbor discovery messages for the address given by the Registrar as being it’s link-local address. The Join Proxy must also advertise this address as the address to which to connect to when advertising it’s existence.

This proxy neighbor discovery means that the pledge will create TCP and UDP connections to the correct Registrar address. This matters as the TCP and UDP pseudo-header checksum includes the destination address, and for the proxy to remain completely stateless, it must not be necessary for the checksum to be updated.

C.4. Use of connected sockets; or IP_PKTINFO for CoAP on Registrar

TCP connections on the registrar SHOULD properly capture the ifindex of the incoming connection into the socket structure. This is normal IPv6 socket API processing. The outgoing responses will go out on the same (virtual) interface by ifindex.

When using UDP sockets with CoAP, the application will have to pay attention to the incoming ifindex on the socket. Access to this information is available using the IP_PKTINFO auxiliary extension which is a standard part of the IPv6 sockets API.

A registrar application could, after receipt of an initial CoAP message from the Pledge, create a connected UDP socket (including the ifindex information). The kernel would then take care of accurate demultiplexing upon receive, and subsequent transmission to the correct interface.

C.5. Use of socket extension rather than virtual interface

Some operating systems on which a Registrar need be implemented may find need for a virtual interface per Join Proxy to be problematic. There are other mechanism which can make be done.

If the IPIP decapsulator can mark the (SYN) packet inside the kernel with the address of the Join Proxy sending the traffic, then an interface per Join Proxy may not be needed. The outgoing path need just pay attention to this extra information and add an appropriate IPIP header on outgoing. A CoAP over UDP mechanism may need to expose this extra information to the application as the UDP sockets are often not connected, and the application will need to specify the outgoing path on each packet send.

Such an additional socket mechanism has not been standardized. Terminating L2TP connections over IPsec transport mode suffers from the same challenges.

Appendix D. MUD Extension

The following extension augments the MUD model to include a single node, as described in [I-D.ietf-opsawg-mud] section 3.6, using the following sample module that has the following tree structure:

```
module: ietf-mud-brski-masa
augment /ietf-mud:mud:
+--rw masa-server?   inet:uri
```

The model is defined as follows:

```
<CODE BEGINS>
module ietf-mud-brski-masa {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-mud-brski-masa";
  prefix ietf-mud-brski-masa;
  import ietf-mud {
    prefix ietf-mud;
  }
  import ietf-inet-types {
    prefix inet;
  }

  organization
    "IETF ANIMA (Autonomic Networking Integrated Model and
    Approach) Working Group";
    contact
    "WG Web: http://tools.ietf.org/wg/anima/
    WG List: anima@ietf.org
    ";
  description
    "BRSKI extension to a MUD file to indicate the
    MASA URL.";

  revision 2017-10-09 {
    description
    "Initial revision.";
    reference
    "RFC XXXX: Manufacturer Usage Description
    Specification";
  }

  augment "/ietf-mud:mud" {
    description
    "BRSKI extension to a MUD file to indicate the
    MASA URL.";
    leaf masa-server {
      type inet:uri;
      description
      "This value is the URI of the MASA server";
    }
  }
}
<CODE ENDS>
```

Appendix E. Example Vouchers

Three entities are involved in a voucher: the MASA issues (signs) it, the registrar's public key is mentioned in the voucher, and the pledge validates it. In order to provide reproduceable examples the public and private keys for an example MASA and Registrar are first listed.

E.1. Keys involved

The Manufacturer has a Certificate Authority that signs the Pledge's IDevID. In addition the Manufacturer's signing authority (the MASA) signs the vouchers, and that certificate must distributed to the devices at manufacturing time so that vouchers can be validated.

E.1.1. MASA key pair for voucher signatures

This private key signs vouchers:

```
-----BEGIN EC PRIVATE KEY-----
MIGkAgEBBDAgiRoYqKoEcfOfvRvmZ5P5Azn58tuI7nSnIy7OgFnCeiNo+BmbgMho
r6lcU60gwVagBwYFK4EEACKhZANiAATZAH3Rb2FvIJOnts+vXuWW35ofyNbCHzjA
zOi2kWZFE1ByurKImNcNMFGirGnRXIXGqWCfw5ICgJ8CuM3vV5ty9bf7KUlOkejz
Tvv+5PV++elkP9HQ83vqTAws2WwWTxI=
-----END EC PRIVATE KEY-----
```

This public key validates vouchers:

```
-----BEGIN CERTIFICATE-----
MIIBzzCCAVagAwIBAgIBATAKBggqhkjOPQQDAjBNMRIwEAYKCZImiZPyLGQBGRYC
Y2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xHDAaBgNVBAMME1Vuc3RydW5n
IEhpZ2h3YXkgQ0EwHhcNMTcwMzI2MTYxOTQwWhcNMTkwMzI2MTYxOTQwWjBHMRIw
EAYKCZImiZPyLGQBGRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xFjAU
BgNVBAMMDVVuc3RydW5nIE1BU0EwdjAQBgcqhkjOPQIBBgUrgQQAIgNiAATZAH3R
b2FvIJOnts+vXuWW35ofyNbCHzjAzOi2kWZFE1ByurKImNcNMFGirGnRXIXGqWCf
w5ICgJ8CuM3vV5ty9bf7KUlOkejzTvv+5PV++elkP9HQ83vqTAws2WwWTxKjEDAO
MAwGA1UdEwEB/wQCMAAwCgYIKoZIzj0EAwIDZwAwZAIwGb0oyM0doP6t3/LSPL5O
DuatEwMYh7WGO+IYTHC8K7EyHBOmCYReKT2+GhV/CLWzAjBNy6UMJTt1tsxJsJqd
MPUIFj+4wZg1AOIb/JoA6M7r33pwLQTrHRxEzVMGfWOkYUw=
-----END CERTIFICATE-----
```

E.1.2. Manufacturer key pair for IDevID signatures

This private key signs IDevID certificates:

```
-----BEGIN EC PRIVATE KEY-----
MIGkAgEBBDAgiRoYqKoEcfOfvRvmZ5P5Azn58tuI7nSnIy7OgFnCeiNo+BmbgMho
r6lcU60gwVagBwYFK4EEACKhZANiAATZAH3Rb2FvIJOnts+vXuWW35ofyNbCHzjA
zOi2kWZFE1ByurKImNcNMFGirGnRXIXGqWCfw5ICgJ8CuM3vV5ty9bf7KUlOkejz
Tvv+5PV++elkP9HQ83vqTAws2WwWTxI=
-----END EC PRIVATE KEY-----
```

This public key validates IDevID certificates:

```
-----BEGIN CERTIFICATE-----
MIIBzzCCAVagAwIBAgIBATAKBggqhkjOPQQDAjBNMRIwEAYKCZImiZPyLGQBGRYC
Y2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xHDAaBgNVBAMME1Vuc3RydW5n
IEhpZ2h3YXkgQ0EwHhcNMTcwMzI2MTYxOTQwWhcNMTkwMzI2MTYxOTQwWjBHMRIw
EAYKCZImiZPyLGQBGRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xFjAU
BgNVBAMMDVVuc3RydW5nIE1BU0EwdjAQBgcqhkjOPQIBBgUrgQQAIgNiAATZAH3R
b2FvIJOnts+vXuWW35ofyNbCHzjAzOi2kWZFE1ByurKImNcNMFGirGnRXIXGqWCf
w5ICgJ8CuM3vV5ty9bf7KUlOkejzTvv+5PV++elkP9HQ83vqTAws2WwWTxKjEDAO
MAwGA1UdEwEB/wQCMAAwCgYIKoZIzj0EAwIDZwAwZAIwGb0oyM0doP6t3/LSPL5O
DuatEwMYh7WGO+IYTHC8K7EyHBOmCYReKT2+GhV/CLWzAjBNy6UMJTt1tsxJsJqd
MPUIFj+4wZg1AOIb/JoA6M7r33pwLQTrHRxEzVMGfWOkYUw=
-----END CERTIFICATE-----
```

E.1.3. Registrar key pair

The registrar key (or chain) is the representative of the domain owner. This key signs Registrar voucher-requests:

```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIF+obiToYYYeMifPsZvrjWJ0yFsCJwIFhpokmT/TULmXoAoGCCqGSM49
AwEHoUQDQgAENWQOzcNMUjP0NrtfeBc0DJLWfeMGgCFdIv6FUz4DifM1ujMBec/g
6W/P6boTmyTGdFOh/8HwKUerL5bpneK8sg==
-----END EC PRIVATE KEY-----
```

The public key is indicated in a pledge voucher-request to show proximity.

```
-----BEGIN CERTIFICATE-----
MIIBrjCCATOgAwIBAgIBAzAKBggqhkjOPQQDAzBOMRIwEAYKCZImiZPyLGQBGRYC
Y2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xHTAbBgNVBAMMFFVuc3RydW5n
IEZvdW50YWluIENBMB4XDTE3MDkwNTAxMTI0NVoXDTE5MDkwNTAxMTI0NVowQzES
MBAGCgmSJomT8ixkARkWAmNhMRkwFwYKCZImiZPyLGQBGRYJc2FuZGVsbWFuMRIw
EAYDVQQDDAlsb2NhbGhvc3QwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAQ1ZA7N
w0xSM/Q2u194FzQMktZ94waAIV0i/oVTPgOJ8zW6MwF5z+Dpb8/puhObJMZ0U6H/
wfApR6svlumd4ryyow0wCzAJBgNVHRMEAjAAMAoGCCqGSM49BAMDA2kAMGYCMQC3
/iTQJ3evYYcgbXhbmzrp64t3QC6qjIeY2jkDx062nuNifVKtyaara3F30AIkKSEC
MQDi29efbTLbdtDk3tecY/rD7V77XaJ6nYCmdDCR54TrSFNLgxvt1lyFM+0fYpYR
c3o=
-----END CERTIFICATE-----
```

The registrar public certificate as decoded by openssl's x509 utility. Note that the registrar certificate is marked with the cmcRA extension.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 3 (0x3)
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: DC=ca, DC=sandelman, CN=Unstrung Fountain CA
        Validity
            Not Before: Sep  5 01:12:45 2017 GMT
            Not After : Sep  5 01:12:45 2019 GMT
        Subject: DC=ca, DC=sandelman, CN=localhost
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
            EC Public Key:
                pub:
                    04:35:64:0e:cd:c3:4c:52:33:f4:36:bb:5f:7
8:17:
                    34:0c:92:d6:7d:e3:06:80:21:5d:22:fe:85:5
3:3e:
                    03:89:f3:35:ba:33:01:79:cf:e0:e9:6f:cf:e
9:ba:
                    13:9b:24:c6:74:53:a1:ff:c1:f0:29:47:ab:2
f:96:
                    e9:9d:e2:bc:b2
                ASN1 OID: prime256v1
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
    Signature Algorithm: ecdsa-with-SHA384
        30:66:02:31:00:b7:fe:24:d0:27:77:af:61:87:20:6d:78:5
b:
        9b:3a:e9:eb:8b:77:40:2e:aa:8c:87:98:da:39:03:c7:4e:b
6:
        9e:e3:62:7d:52:ad:c9:a6:ab:6b:71:77:d0:02:24:29:21:0
2:
        31:00:e2:db:d7:9f:6d:32:db:76:d0:e4:de:d7:9c:63:fa:c
3:
        ed:5e:fb:5d:a2:7a:9d:80:a6:74:30:91:e7:84:eb:48:53:4
b:
        83:1b:ed:d6:5c:85:33:ed:1f:62:96:11:73:7a
```

E.1.4. Pledge key pair

The pledge has an IDevID key pair built in at manufacturing time:

```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIL+ue8PQcN+M7LFBGPsompYwobI/rsoHnTb2a+0hO+8joAoGCCqGSM49
AwEHoUQDQgAEumBVaDlX87WyME8CJToyt9NWy6sYw0DTbjjJIn79pgr7ALa//Y8p
r70WpK1SIaiUeeFw7e+lCzTp1Z+wJu14Bg==
-----END EC PRIVATE KEY-----
```

The public key is used by the registrar to find the MASA. The MASA URL is in an extension described in Section 2.3. RFC-EDITOR: Note that these certificates are using a Private Enterprise Number for the not-yet-assigned by IANA MASA URL, and need to be replaced before AUTH48.

```
-----BEGIN CERTIFICATE-----
MIICMjCCAbegAwIBAgIBDDAKBggqhkjOPQQDAjBNMRIwEAYKCZImiZPyLGQBGRYC
Y2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xHDAaBgNVBAMME1Vuc3RydW5n
IEhpZ2h3YXkgQ0EwIBcNMTcxMDEyMTM1MjUyWhgPMjk5OTEyMzEwMDAwMDBaMEsx
EjAQBgoJkiaJk/IsZAEZFgJjYTEZMBcGCgmSJomT8ixkARkWCXNhbmRlbG1hbjEa
MBgGA1UEAwwRMDAtRDAtRTUtRjItMDAtMDIwWTATBgcqhkjOPQIBBggqhkjOPQMB
BwNCAARJp5i0dU1aUnR2u8wMRwgkNupNbNM7m1n0mj+0KJZjcPIqID+trPjTSobt
uIdpRPfGZ8hU/nIUveqwyoYI8BPbo4GHMIGEMB0GA1UdDgQWBBQdMRZhthFQmzz6
E7YVXzkL7XZDKjAJBgNVHRMEAjAAMCsGA1UdEQQkMCKgIAYJKwYBBAGC7lIBoBMM
ETAwLUQwLUU1LUYyLTAwLTAyMCsGCSsGAQQBgu5SAgQeDBxodHRwczovL2hpZ2h3
YXkuc2FuZGVsbWFuLmNhMAoGCCqGSM49BAMCA2kAMGYCMQDhJ1N+eanW1U/e5qoM
SGvUvWHR7uic8cJbh7vXy580nBs8bpNn60k/+IzvEUetMzICMQCr1uxvdYeKq7mb
RXCR4ZCJsw67fJ7jyXZbCUSir+3wBT2+lWggzPDRgYB5ABb7sAw=
-----END CERTIFICATE-----
```

The pledge public certificate as decoded by openssl’s x509 utility so that the extensions can be seen. A second custom Extension is included to provided to contain the EUI48/EUI64 that the pledge will configure.

```
   Certificate:
       Data:
           Version: 3 (0x2)
           Serial Number: 12 (0xc)
           Signature Algorithm: ecdsa-with-SHA256
           Issuer: DC=ca, DC=sandelman, CN=Unstrung Highway CA
           Validity
               Not Before: Oct 12 13:52:52 2017 GMT
               Not After : Dec 31 00:00:00 2999 GMT
           Subject: DC=ca, DC=sandelman, CN=00-D0-E5-F2-00-02
           Subject Public Key Info:
               Public Key Algorithm: id-ecPublicKey
               EC Public Key:
                   pub:
                       04:49:a7:98:b4:75:4d:5a:52:74:76:bb:cc:0
   c:47:
                       08:24:36:ea:4d:6c:d3:3b:9b:59:f4:9a:3f:b
   4:28:
                       96:63:70:f2:2a:20:3f:ad:ac:f8:d3:4a:86:e
   d:b8:
                       87:69:44:f7:c6:67:c8:54:fe:72:14:bd:ea:b
   0:ca:
                       86:08:f0:13:db
                   ASN1 OID: prime256v1
           X509v3 extensions:
               X509v3 Subject Key Identifier:
                   1D:31:16:61:B6:11:50:9B:3C:FA:13:B6:15:5F:39
   :0B:ED:76:43:2A
               X509v3 Basic Constraints:
                   CA:FALSE
               X509v3 Subject Alternative Name:
                   othername:<unsupported>
               1.3.6.1.4.1.46930.2:
                   ..https://highway.sandelman.ca
       Signature Algorithm: ecdsa-with-SHA256
           30:66:02:31:00:e1:27:53:7e:79:a9:d6:d5:4f:de:e6:aa:0
   c:
           48:6b:d4:bd:61:d1:ee:e8:9c:f1:c2:5b:87:bb:d7:cb:9f:3
   4:
           9c:1b:3c:6e:93:67:eb:49:3f:f8:8c:ef:11:47:ad:33:32:0
   2:
           31:00:ab:d6:ec:6f:75:87:8a:ab:b9:9b:45:70:91:e1:90:8
   9:
           b3:0e:bb:7c:9e:e3:c9:76:5b:09:44:a2:af:ed:f0:05:3d:b
   e:
           95:68:20:cc:f0:d1:81:80:79:00:16:fb:b0:0c
```

E.2. Example process

RFC-EDITOR: these examples will need to be replaced with CMS versions once IANA has assigned the eContentType in [I-D.ietf-anima-voucher].

E.2.1. Pledge to Registrar

As described in Section 5.2, the pledge will sign a pledge voucher-request containing the Registrar's public key in the proximity-registrar-cert field. The base64 has been wrapped at 60 characters for presentation reasons.

MIIHHAYJKoZIhvcNAQcCoIIHDTCCBwkCAQExDzANBglghkgBZQMEAgEFADCC
Aw4GCSqGSIb3DQEHAaCCAv8EggL7eyJpZXRmLXZvdWNoZXItcmVxdWVzdDp2
b3VjaGVyIjp7ImFzc2VydGlvbiI6InByb3hpbWl0eSIsImNyZWF0ZWQtb24i
OiIyMDE3LTA5LTAxIiwic2VyaWFsLW51bWJlciI6IjAwLUQwLUU1LUYyLTAw
LTAyIiwibm9uY2UiOiJEc3M5OXNCcjNwTk1PQUNlLUxZWTd3IiwicHJveGlt
aXR5LXJlZ2lzdHJhci1jZXJ0IjoiTUlJQnJqQ0NBVE9nQXdJQkFnSUJBekFL
QmdncWhrak9QUVFEQXpCT01SSXdFQVlLQ1pJbWlaUHlMR1FCR1JZQ1kyRXhH
VEFYQmdvSmtpYUprL0lzWkFFWkZnbHpZVzVrWld4dFlXNHhIVEFiQmdOVkJB
TU1GRlZ1YzNSeWRXNW5JRVp2ZFc1MFlXbHVJRU5CTUI0WERURTNNRGt3TlRB
eE1USTBOVm9YRFRFNU1Ea3dOVEF4TVRJME5Wb3dRekVTTUJBR0NnbVNKb21U
OGl4a0FSa1dBbU5oTVJrd0Z3WUtDWkltaVpQeUxHUUJHUllKYzJGdVpHVnNi
V0Z1TVJJd0VBWURWUVFEREFsc2IyTmhiR2h2YzNRd1dUQVRCZ2NxaGtqT1BR
SUJCZ2dxaGtqT1BRTUJCd05DQUFRMVpBN053MHhTTS9RMnUxOTRGelFNa3Ra
OTR3YUFJVjBpL29WVFBnT0o4elc2TXdGNXorRHBiOC9wdWhPYkpNWjBVNkgv
d2ZBcFI2c3ZsdW1kNHJ5eW93MHdDekFKQmdOVkhSTUVBakFBTUFvR0NDcUdT
TTQ5QkFNREEya0FNR1lDTVFDMy9pVFFKM2V2WVljZ2JYaGJtenJwNjR0M1FD
NnFqSWVZMmprRHgwNjJudU5pZlZLdHlhYXJhM0YzMEFJa0tTRUNNUURpMjll
ZmJUTGJkdERrM3RlY1kvckQ3Vjc3WGFKNm5ZQ21kRENSNTRUclNGTkxneHZ0
MWx5Rk0rMGZZcFlSYzNvPSJ9faCCAjYwggIyMIIBt6ADAgECAgEMMAoGCCqG
SM49BAMCME0xEjAQBgoJkiaJk/IsZAEZFgJjYTEZMBcGCgmSJomT8ixkARkW
CXNhbmRlbG1hbjEcMBoGA1UEAwwTVW5zdHJ1bmcgSGlnaHdheSBDQTAgFw0x
NzEwMTIxMzUyNTJaGA8yOTk5MTIzMTAwMDAwMFowSzESMBAGCgmSJomT8ixk
ARkWAmNhMRkwFwYKCZImiZPyLGQBGRYJc2FuZGVsbWFuMRowGAYDVQQDDBEw
MC1EMC1FNS1GMi0wMC0wMjBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABEmn
mLR1TVpSdHa7zAxHCCQ26k1s0zubWfSaP7QolmNw8iogP62s+NNKhu24h2lE
98ZnyFT+chS96rDKhgjwE9ujgYcwgYQwHQYDVR0OBBYEFB0xFmG2EVCbPPoT
thVfOQvtdkMqMAkGA1UdEwQCMAAwKwYDVR0RBCQwIqAgBgkrBgEEAYLuUgGg
EwwRMDAtRDAtRTUtRjItMDAtMDIwKwYJKwYBBAGC7lICBB4MHGh0dHBzOi8v
aGlnaHdheS5zYW5kZWxtYW4uY2EwCgYIKoZIzj0EAwIDaQAwZgIxAOEnU355
qdbVT97mqgxIa9S9YdHu6JzxwluHu9fLnzScGzxuk2frST/4jO8RR60zMgIx
AKvW7G91h4qruZtFcJHhkImzDrt8nuPJdlsJRKKv7fAFPb6VaCDM8NGBgHkA
FvuwDDGCAaUwggGhAgEBMFIwTTESMBAGCgmSJomT8ixkARkWAmNhMRkwFwYK
CZImiZPyLGQBGRYJc2FuZGVsbWFuMRwwGgYDVQQDDBNVbnN0cnVuZyBIaWdo
d2F5IENBAgEMMA0GCWCGSAFlAwQCAQUAoIHkMBgGCSqGSIb3DQEJAzELBgkq
hkiG9w0BBwEwHAYJKoZIhvcNAQkFMQ8XDTE3MTAxMjE3NTQzMFowLwYJKoZI
hvcNAQkEMSIEIP59cuKVAPkKOOlQIaIV/W1AsWKbmVmBd9wFSuD5yLafMHkG
CSqGSIb3DQEJDzFsMGowCwYJYIZIAWUDBAEqMAsGCWCGSAFlAwQBFjALBglg
hkgBZQMEAQIwCgYIKoZIhvcNAwcwDgYIKoZIhvcNAwICAgCAMA0GCCqGSIb3
DQMCAgFAMAcGBSsOAwIHMA0GCCqGSIb3DQMCAgEoMAoGCCqGSM49BAMCBEYw
RAIgYUy0NTdP+xTkm/Et69eI++S/2z3dQwPKOwdL0cDCSvACIAh3jJbybMnK
cf7DKKnsn2G/O06HeB/8imMI+hnA7CfN

file: examples/vr_00-D0-E5-F2-00-02.pkcs

The ASN1 decoding of the artifact:

The JSON contained in the voucher request:

E.2.2. Registrar to MASA

As described in Section 5.4 the Registrar will sign a registrar voucher-request, and will include pledge's voucher request in the prior-signed-voucher-request.

MIIN2gYJKoZIhvcNAQcCoIINyzCCDccCAQExDzANBglghkgBZQMEAgEFADCC
Ck4GCSqGSIb3DQEHAaCCCj8Eggo7eyJpZXRmLXZvdWNoZXItcmVxdWVzdDp2
b3VjaGVyIjp7ImFzc2VydGlvbiI6InByb3hpbWl0eSIsImNyZWF0ZWQtb24i
OiIyMDE3LTA5LTE1VDAwOjAwOjAwLjAwMFoiLCJzZXJpYWwtbnVtYmVyIjoi
SkFEQTEyMzQ1Njc4OSIsIm5vbmNlIjoiYWJjZDEyMzQiLCJwcmlvci1zaWdu
ZWQtdm91Y2hlci1yZXF1ZXN0IjoiTUlJSEhRWUpLb1pJaHZjTkFRY0NvSUlI
RGpDQ0J3b0NBUUV4RHpBTkJnbGdoa2dCWlFNRUFnRUZBRENDQXc0R0NTcUdT
SWIzRFFFSEFhQ0NBdjhFZ2dMN2V5SnBaWFJtTFhadmRXTm9aWEl0Y21WeGRX
VnpkRHAyYjNWamFHVnlJanA3SW1GemMyVnlkR2x2YmlJNkluQnliM2hwYlds
MGVTSXNJbU55WldGMFpXUXRiMjRpT2lJeU1ERTNMVEE1TFRBeElpd2ljMlZ5
YVdGc0xXNTFiV0psY2lJNklqQXdMVVF3TFVVMUxVWXlMVEF3TFRBeUlpd2li
bTl1WTJVaU9pSkVjM001T1hOQ2NqTndUazFQUVVObExVeFpXVGQzSWl3aWNI
SnZlR2x0YVhSNUxYSmxaMmx6ZEhKaGNpMWpaWEowSWpvaVRVbEpRbkpxUTBO
QlZFOW5RWGRKUWtGblNVSkJla0ZMUW1kbmNXaHJhazlRVVZGRVFYcENUMDFT
U1hkRlFWbExRMXBKYldsYVVIbE1SMUZDUjFKWlExa3lSWGhIVkVGWVFtZHZT
bXRwWVVwckwwbHpXa0ZGV2tabmJIcFpWelZyV2xkNGRGbFhOSGhJVkVGaVFt
ZE9Wa0pCVFUxR1JsWjFZek5TZVdSWE5XNUpSVnAyWkZjMU1GbFhiSFZKUlU1
Q1RVSTBXRVJVUlROTlJHdDNUbFJCZUUxVVNUQk9WbTlZUkZSRk5VMUVhM2RP
VkVGNFRWUkpNRTVXYjNkUmVrVlRUVUpCUjBObmJWTktiMjFVT0dsNGEwRlNh
MWRCYlU1b1RWSnJkMFozV1V0RFdrbHRhVnBRZVV4SFVVSkhVbGxLWXpKR2RW
cEhWbk5pVjBaMVRWSkpkMFZCV1VSV1VWRkVSRUZzYzJJeVRtaGlSMmgyWXpO
UmQxZFVRVlJDWjJOeGFHdHFUMUJSU1VKQ1oyZHhhR3RxVDFCUlRVSkNkMDVE
UVVGUk1WcEJOMDUzTUhoVFRTOVJNblV4T1RSR2VsRk5hM1JhT1RSM1lVRkpW
akJwTDI5V1ZGQm5UMG80ZWxjMlRYZEdOWG9yUkhCaU9DOXdkV2hQWWtwTldq
QlZOa2d2ZDJaQmNGSTJjM1pzZFcxa05ISjVlVzkzTUhkRGVrRktbWRPVmto
U1RVVkJha0ZCVFVGdlIwTkRjVWRUVFRRNVFrRk5SRUV5YTBGTlIxbERUVkZE
TXk5cFZGRktNMlYyV1ZsaloySllhR0p0ZW5Kd05qUjBNMUZETm5GcVNXVlpN
bXByUkhnd05qSnVkVTVwWmxaTGRIbGhZWEpoTTBZek1FRkphMHRUUlVOTlVV
UnBNamxsWm1KVVRHSmtkRVJyTTNSbFkxa3Zja1EzVmpjM1dHRktObTVaUTIx
a1JFTlNOVFJVY2xOR1RreG5lSFowTVd4NVJrMHJNR1paY0ZsU1l6TnZQU0o5
ZmFDQ0FqWXdnZ0l5TUlJQnQ2QURBZ0VDQWdFTU1Bb0dDQ3FHU000OUJBTUNN
RTB4RWpBUUJnb0praWFKay9Jc1pBRVpGZ0pqWVRFWk1CY0dDZ21TSm9tVDhp
eGtBUmtXQ1hOaGJtUmxiRzFoYmpFY01Cb0dBMVVFQXd3VFZXNXpkSEoxYm1j
Z1NHbG5hSGRoZVNCRFFUQWdGdzB4TnpFd01USXhNelV5TlRKYUdBOHlPVGs1
TVRJek1UQXdNREF3TUZvd1N6RVNNQkFHQ2dtU0pvbVQ4aXhrQVJrV0FtTmhN
Umt3RndZS0NaSW1pWlB5TEdRQkdSWUpjMkZ1WkdWc2JXRnVNUm93R0FZRFZR
UUREQkV3TUMxRU1DMUZOUzFHTWkwd01DMHdNakJaTUJNR0J5cUdTTTQ5QWdF
R0NDcUdTTTQ5QXdFSEEwSUFCRW1ubUxSMVRWcFNkSGE3ekF4SENDUTI2azFz
MHp1YldmU2FQN1FvbG1Odzhpb2dQNjJzK05OS2h1MjRoMmxFOThabnlGVCtj
aFM5NnJES2hnandFOXVqZ1ljd2dZUXdIUVlEVlIwT0JCWUVGQjB4Rm1HMkVW
Q2JQUG9UdGhWZk9RdnRka01xTUFrR0ExVWRFd1FDTUFBd0t3WURWUjBSQkNR
d0lxQWdCZ2tyQmdFRUFZTHVVZ0dnRXd3Uk1EQXRSREF0UlRVdFJqSXRNREF0

TURJd0t3WUpLd1lCQkFHQzdsSUNCQjRNSEdoMGRIQnpPaTh2YUdsbmFIZGhl
UzV6WVc1a1pXeHRZVzR1WTJFd0NnWUlLb1pJemowRUF3SURhUUF3WmdJeEFP
RW5VMzU1cWRiVlQ5N21xZ3hJYTlTOVlkSHU2Snp4d2x1SHU5ZkxuelNjR3p4
dWsyZnJTVC80ak84UlI2MHpNZ0l4QUt2VzdHOTFoNHFydVp0RmNKSGhrSW16
RHJ0OG51UEpkbHNKUktLdjdmQUZQYjZWYUNETThOR0JnSGtBRnZ1d0RER0NB
YVl3Z2dHaUFnRUJNRkl3VFRFU01CQUdDZ21TSm9tVDhpeGtBUmtXQW1OaE1S
a3dGd1lLQ1pJbWlaUHlMR1FCR1JZSmMyRnVaR1ZzYldGdU1Sd3dHZ1lEVlFR
RERCTlZibk4wY25WdVp5QklhV2RvZDJGNUlFTkJBZ0VNTUEwR0NXQ0dTQUZs
QXdRQ0FRVUFvSUhrTUJnR0NTcUdTSWIzRFFFSkF6RUxCZ2txaGtpRzl3MEJC
d0V3SEFZSktvWklodmNOQVFrRk1ROFhEVEUzTVRBeE1qRXpOVGd5TTFvd0x3
WUpLb1pJaHZjTkFRa0VNU0lFSVA1OWN1S1ZBUGtLT09sUUlhSVYvVzFBc1dL
Ym1WbUJkOXdGU3VENXlMYWZNSGtHQ1NxR1NJYjNEUUVKRHpGc01Hb3dDd1lK
WUlaSUFXVURCQUVxTUFzR0NXQ0dTQUZsQXdRQkZqQUxCZ2xnaGtnQlpRTUVB
UUl3Q2dZSUtvWklodmNOQXdjd0RnWUlLb1pJaHZjTkF3SUNBZ0NBTUEwR0ND
cUdTSWIzRFFNQ0FnRkFNQWNHQlNzT0F3SUhNQTBHQ0NxR1NJYjNEUU1DQWdF
b01Bb0dDQ3FHU000OUJBTUNCRWN3UlFJZ0VNZzFkSkw3RmNkdHJWRHg4cUNh
em9lOSsyMk56NFp3UkI5Z0FUR0w3TU1DSVFEanNzVWxaekpxcDIva0NkNFdo
eFVoc2FDcFRGd1Bybk5ldzV3Q2tZVUY4UT09In19oIIBsjCCAa4wggEzoAMC
AQICAQMwCgYIKoZIzj0EAwMwTjESMBAGCgmSJomT8ixkARkWAmNhMRkwFwYK
CZImiZPyLGQBGRYJc2FuZGVsbWFuMR0wGwYDVQQDDBRVbnN0cnVuZyBGb3Vu
dGFpbiBDQTAeFw0xNzA5MDUwMTEyNDVaFw0xOTA5MDUwMTEyNDVaMEMxEjAQ
BgoJkiaJk/IsZAEZFgJjYTEZMBcGCgmSJomT8ixkARkWCXNhbmRlbG1hbjES
MBAGA1UEAwwJbG9jYWxob3N0MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE
NWQOzcNMUjP0NrtfeBc0DJLWfeMGgCFdIv6FUz4DifM1ujMBec/g6W/P6boT
myTGdFOh/8HwKUerL5bpneK8sqMNMAswCQYDVR0TBAIwADAKBggqhkjOPQQD
AwNpADBmAjEAt/4k0Cd3r2GHIG14W5s66euLd0AuqoyHmNo5A8dOtp7jYn1S
rcmmq2txd9ACJCkhAjEA4tvXn20y23bQ5N7XnGP6w+1e+12iep2ApnQwkeeE
60hTS4Mb7dZchTPtH2KWEXN6MYIBpzCCAaMCAQEwUzBOMRIwEAYKCZImiZPy
LGQBGRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xHTAbBgNVBAMM
FFVuc3RydW5nIEZvdW50YWluIENBAgEDMA0GCWCGSAFlAwQCAQUAoIHkMBgG
CSqGSIb3DQEJAzELBgkqhkiG9w0BBwEwHAYJKoZIhvcNAQkFMQ8XDTE3MTAy
NjAxMzYxOFowLwYJKoZIhvcNAQkEMSIEIEQBM73PZzPo7tE9Mj8gQvaaYeMQ
OsxlACaW/HenAqNwMHkGCSqGSIb3DQEJDzFsMGowCwYJYIZIAWUDBAEqMAsG
CWCGSAFlAwQBFjALBglghkgBZQMEAQIwCgYIKoZIhvcNAwcwDgYIKoZIhvcN
AwICAgCAMA0GCCqGSIb3DQMCAgFAMAcGBSsOAwIHMA0GCCqGSIb3DQMCAgEo
MAoGCCqGSM49BAMCBEcwRQIgDdp5uPUlMKp7GFQAD7ypAgqFv8q+KkJt6c3O
7iVpVI8CIQCD1u8BkxipvigwvIDmWfjlYdJxcvozNjffq5j3UHg7Rg==

file: examples/parboiled_vr_00-D0-E5-F2-00-02.pkcs

The ASN1 decoding of the artifact:

E.2.3. MASA to Registrar

The MASA will return a voucher to the Registrar, to be relayed to the pledge.

MIIG3AYJKoZIhvcNAQcCoIIGzTCCBskCAQExDzANBglghkgBZQMEAgEFADCC
AxAGCSqGSIb3DQEHAaCCAwEEggL9eyJpZXRmLXZvdWNoZXI6dm91Y2hlciI6
eyJhc3NlcnRpb24iOiJsb2dnZWQiLCJjcmVhdGVkLW9uIjoiMjAxNy0xMC0x
MlQxMzo1NDozMS40MzktMDQ6MDAiLCJzZXJpYWwtbnVtYmVyIjoiMDAtRDAt
RTUtRjItMDAtMDIiLCJub25jZSI6IkRzczk5c0JyM3BOTU9BQ2UtTFlZN3ci
LCJwaW5uZWQtZG9tYWluLWNlcnQiOiJNSUlCcmpDQ0FUT2dBd0lCQWdJQkF6
QUtCZ2dxaGtqT1BRUURBekJPTVJJd0VBWUtDWkltaVpQeUxHUUJHUllDWTJF
eEdUQVhCZ29Ka2lhSmsvSXNaQUVaRmdsellXNWtaV3h0WVc0eEhUQWJCZ05W
QkFNTUZGVnVjM1J5ZFc1bklFWnZkVzUwWVdsdUlFTkJNQjRYRFRFM01Ea3dO
VEF4TVRJME5Wb1hEVEU1TURrd05UQXhNVEkwTlZvd1F6RVNNQkFHQ2dtU0pv
bVQ4aXhrQVJrV0FtTmhNUmt3RndZS0NaSW1pWlB5TEdRQkdSWUpjMkZ1WkdW
c2JXRnVNUkl3RUFZRFZRUUREQWxzYjJOaGJHaHZjM1F3V1RBVEJnY3Foa2pP
UFFJQkJnZ3Foa2pPUFFNQkJ3TkNBQVExWkE3TncweFNNL1EydTE5NEZ6UU1r
dFo5NHdhQUlWMGkvb1ZUUGdPSjh6VzZNd0Y1eitEcGI4L3B1aE9iSk1aMFU2
SC93ZkFwUjZzdmx1bWQ0cnl5b3cwd0N6QUpCZ05WSFJNRUFqQUFNQW9HQ0Nx
R1NNNDlCQU1EQTJrQU1HWUNNUUMzL2lUUUozZXZZWWNnYlhoYm16cnA2NHQz
UUM2cWpJZVkyamtEeDA2Mm51TmlmVkt0eWFhcmEzRjMwQUlrS1NFQ01RRGky
OWVmYlRMYmR0RGszdGVjWS9yRDdWNzdYYUo2bllDbWREQ1I1NFRyU0ZOTGd4
dnQxbHlGTSswZllwWVJjM289In19oIIB0zCCAc8wggFWoAMCAQICAQEwCgYI
KoZIzj0EAwIwTTESMBAGCgmSJomT8ixkARkWAmNhMRkwFwYKCZImiZPyLGQB
GRYJc2FuZGVsbWFuMRwwGgYDVQQDDBNVbnN0cnVuZyBIaWdod2F5IENBMB4X
DTE3MDMyNjE2MTk0MFoXDTE5MDMyNjE2MTk0MFowRzESMBAGCgmSJomT8ixk
ARkWAmNhMRkwFwYKCZImiZPyLGQBGRYJc2FuZGVsbWFuMRYwFAYDVQQDDA1V
bnN0cnVuZyBNQVNBMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAE2QB90W9hbyCT
p7bPr17llt+aH8jWwh84wMzotpFmRRNQcrqyiJjXDTBRoqxp0VyFxqlgn8OS
AoCfArjN71ebcvW3+ylJTpHo8077/uT1fvnpZD/R0PN76kwMLNlsFk8SoxAw
DjAMBgNVHRMBAf8EAjAAMAoGCCqGSM49BAMCA2cAMGQCMBm9KMjNHaD+rd/y
0jy+Tg7mrRMDGIe1hjviGExwvCuxMhwTpgmEXik9vhoVfwi1swIwTculDCU7
dbbMSbCanTD1CBY/uMGYNQDiG/yaAOjO6996cC0E6x0cRM1TBn1jpGFMMYIB
xjCCAcICAQEwUjBNMRIwEAYKCZImiZPyLGQBGRYCY2ExGTAXBgoJkiaJk/Is
ZAEZFglzYW5kZWxtYW4xHDAaBgNVBAMME1Vuc3RydW5nIEhpZ2h3YXkgQ0EC
AQEwDQYJYIZIAWUDBAIBBQCggeQwGAYJKoZIhvcNAQkDMQsGCSqGSIb3DQEH
ATAcBgkqhkiG9w0BCQUxDxcNMTcxMDEyMTc1NDMxWjAvBgkqhkiG9w0BCQQx
IgQgQXnG628cIW8MoYfB1ljDDlLlJQlxED2tnjcvkLEfix0weQYJKoZIhvcN
AQkPMWwwajALBglghkgBZQMEASowCwYJYIZIAWUDBAEWMAsGCWCGSAFlAwQB
AjAKBggqhkiG9w0DBzAOBggqhkiG9w0DAgICAIAwDQYIKoZIhvcNAwICAUAw
BwYFKw4DAgcwDQYIKoZIhvcNAwICASgwCgYIKoZIzj0EAwIEZzBlAjEAhzid
/AkNjttpSP1rflNppdHsi324Z2+TXJxueewnJ8z/2NXb+Tf3DsThv7du00Oz
AjBjyOnmkkSKHsPR2JluA5c6wovuPEnNKP32daGGeFKGEHMkTInbrqipC881
/5K9Q+k=

file: examples/voucher_00-D0-E5-F2-00-02.pkcs

The ASN1 decoding of the artifact:

Authors' Addresses

Max Pritikin
Cisco

Email: pritikin@cisco.com

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca
URI: http://www.sandelman.ca/

Michael H. Behringer
Cisco

Email: mbehring@cisco.com

Steinthor Bjarnason
Arbor Networks

Email: sbjarnason@arbor.net

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

ANIMA
Internet-Draft
Intended status: Informational
Expires: April 22, 2018

M. Behringer, Ed.
B. Carpenter
Univ. of Auckland
T. Eckert
Futurewei Technologies Inc.
L. Ciavaglia
P. Peloso
Nokia
B. Liu
Huawei Technologies
J. Nobre
Federal University of Rio Grande do Sul
J. Strassner
Huawei Technologies

October 19, 2017

A Reference Model for Autonomic Networking
draft-ietf-anima-reference-model-05

Abstract

This document describes a reference model for Autonomic Networking. The goal is to define how the various elements in an autonomic context work 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.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 22, 2018.

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. While the document is written as generally as possible, the initial solutions are limited to the chartered scope of the WG.

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.

This document describes phase 1 of an Autonomic Networking solution, and covers primarily the WG items of the ANIMA WG as of July 2017. Sections marked with (*) do not represent chartered items at this time. New WG items will require an update to this document, or potentially a new document.

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.

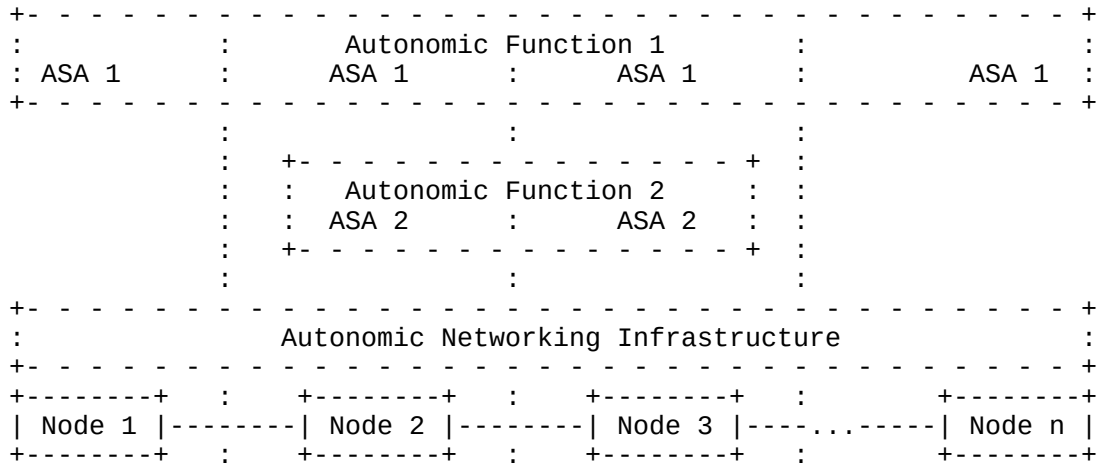


Figure 1: High level view of an Autonomic Network

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.

The Autonomic Networking Infrastructure (ANI) therefore is the foundation for autonomic functions. The current charter of the ANIMA WG is to specify the ANI, using a few autonomic functions as use cases.

3. The Autonomic Network Element

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, 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.

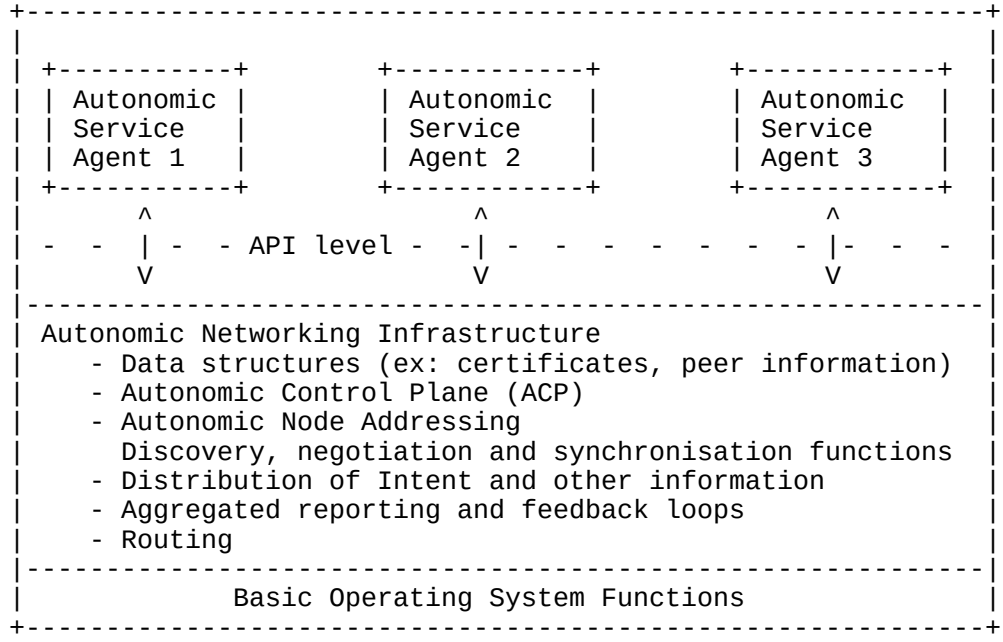


Figure 2: Model of an autonomic node

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). The Autonomic Control Plane (ACP) is the summary of all interactions of the Autonomic Networking Infrastructure with other nodes and 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, 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 Networking Infrastructure (ANI) is normally built 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 of the adjacent autonomic node’s certificate, although all autonomic interactions must verify validity and trust independently.

The adjacency table is fed by the following inputs:

- 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. This is described in [I-D.ietf-anima-grasp].

- Vendor re-direct: A new device may receive information on where its home network is through a vendor based MASA re-direct; this is typically a routable address. See [I-D.ietf-anima-bootstrapping-keyinfra].

- 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:

- 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 vendor MASA re-direct, which will be entered into the adjacency table (see second bullet above). See [I-D.ietf-anima-bootstrapping-keyinfra].

- 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].

- Once the node is part of the ACP of a domain, it will use GRASP discovery [I-D.ietf-anima-grasp] to find Registrar(s) of its domain and potentially other services.

- If the node is part of an ACP and has discovered via GRASP at least one Registrar in its domain, it will start the "join assistant" ASA, and act as a join assistant for neighboring nodes that need to be bootstrapped. See [I-D.ietf-anima-bootstrapping-keyinfra].

- 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.

3.3.1. State 1: Factory Default

An autonomic node is leaving 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:

- 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.

3.3.2. State 2: Enrolled

An autonomic node is in the state "enrolled" if it has a domain identity (LDevID). 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:

- 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.

- Factory reset: A factory reset removes all configuration and the domain identity (LDevID) from the device. Next state: 1.

3.3.3. State 3: In ACP

In this state, the autonomic node has at least one ACP channel to another device. It can participate in further autonomic transactions, such as starting autonomic service agents. For example 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:

- Leaving ACP: The device drops the last (or only) ACP channel to an adjacent device. Next state: 2.

- Factory reset: A factory reset removes all configuration and the domain identity (LDevID) from the device. Next state: 1.

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.

Out of scope are addressing approaches for the data plane of the network, which 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 Nodes may support Autonomic Service Agents in different virtual machines or containers. The addressing scheme should support this architecture.

- Simplicity: To make engineering simpler, and to give the human administrator an easy way to trouble-shoot autonomic functions.

- o Scale: The proposed scheme should work in any network of any size.

- o 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 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 totality of autonomic interactions forms the "Autonomic Control Plane". This control plane can be either implemented in the global routing table of a node, such as IGPs in today’s networks; or it can be provided as an overlay network. The document "An Autonomic Control Plane" ([I-D.ietf-anima-autonomic-control-plane]) describes the details.

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.

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.

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 to prove its identity. [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 be multi-threaded so that it can 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.

Thus we can distinguish at least three classes of ASAs:

- Simple ASAs with a small footprint that could run anywhere.
- Complex, 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, will be self-aware. Every autonomic node will be loaded with various functions and ASAs and will be aware of its own capabilities, typically decided by the hardware, firmware or pre-installed software. Its exact role may depend on Intent and on the surrounding network behaviors, which may include forwarding behaviors, aggregation properties, topology location, bandwidth, tunnel or translation properties, etc. The surrounding topology will depend on the network planning. Following an initial discovery phase, the device properties and those of its neighbors are the foundation of the behavior of a specific device. A device 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.liu-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 recovery actions, with a complete restart of the ASA as a last resort. 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. Whichever the ASA, its management and its interactions with the ANI must 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 host 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:

- 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

- Control of ASA control-loop during the operation: a signaling has to carry formatted messages to control ASA execution (at least starting and stopping 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.

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:

- Autonomic functions can use traditional methods and protocols (e.g., SNMP and NETCONF) to perform management tasks, inside and outside the ACP;
- Autonomic functions can conflict with behavior enforced by the same traditional methods and protocols;
- 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 by the ANIMA charter at the time of this writing. This section is for informational purposes only.

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 SUPA) 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 (*)

At the time of this writing, aggregated reporting is not in the ANIMA charter. This section is provided for information only.

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].

Several 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 advertisement 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 level which is described in the autonomic Intent.

Reporting in an autonomic network may be in the same abstraction level of the Intent. In this context, the visibility on 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 can be prompted with a default action, and has the possibility to acknowledge or override the proposed default action.

7.5. Control Loops (*)

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 (*)

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:

- pre-conditions that must be satisfied before the method can start execution
- post-conditions that must be satisfied when the method has finished execution
- invariant attributes that must not change during the execution of the method

7.7. Data Model (*)

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 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 (*)

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.

- 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:

- 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:

- o protected against modification.

- o authenticated.

- o protected against replay attacks.

- o encrypted.

- o 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 not protected 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, but this would be quite limited (for example, this would be highly unlikely to reveal any specific information about user traffic).

9.2. Risk of Insider Attacks

An autonomic network consists of autonomic devices that form a distributed self-managing system. Devices within a domain share a common trust anchor and thus implicitly trust each other. 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:

- o 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.

- o 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.

- o 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.

12. References

12.1. Normative References

[I-D.ietf-anima-autonomic-control-plane]
Behringer, M., Eckert, T., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-ietf-anima-autonomic-control-plane-10 (work in progress), September 2017.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-07 (work in progress), July 2017.

[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.

12.2. Informative References

[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-prefix-management]
Jiang, S., Du, Z., Carpenter, B., and Q. Sun, "Autonomic IPv6 Edge Prefix Management in Large-scale Networks", draft-ietf-anima-prefix-management-05 (work in progress), August 2017.

[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-api]
Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)", draft-liu-anima-grasp-api-05 (work in progress), October 2017.

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

[IDevID] IEEE Standard, , "IEEE 802.1AR Secure Device Identifier", December 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

[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

Peloso Pierre
Nokia
Villarceaux
Nozay 91460
FR

Email: pierre.peloso@nokia.com

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

Email: leo.liubing@huawei.com

Jeferson Campos Nobre
Federal University of Rio Grande do Sul
Av. Bento Goncalves, 9500
Porto Alegre 91501-970
Brazil

Email: jcnobre@inf.ufrgs.br

John Strassner
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: john.sc.strassner@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 5, 2018

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

October 2, 2017

Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)
draft-liu-anima-grasp-api-05

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 5, 2018.

Copyright Notice

Table of Contents

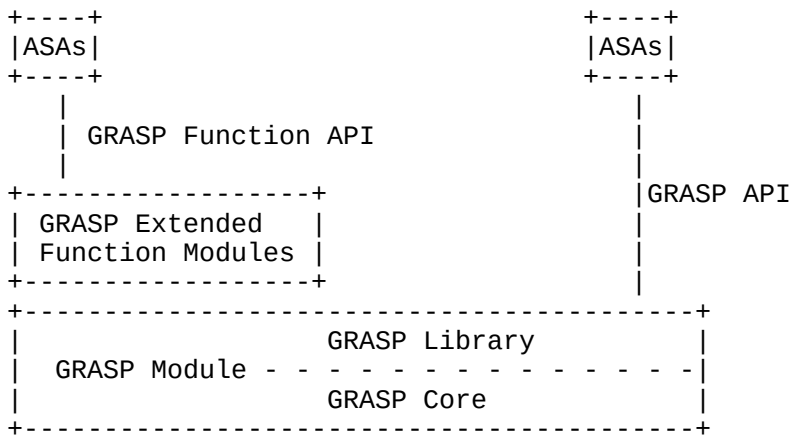
1. Introduction

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

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

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

that provides the API and the other being core code containing common components such as multicast handling and the discovery cache. The details of this are system-dependent.



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

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

2. GRASP API for ASA

2.1. Design Principles

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

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

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

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

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

This is a preliminary version. Two particular gaps exist:

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

2.2. API definition

2.2.1. Parameters and data structures

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

2.2.1.1. Errorcode

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

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

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

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

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

2.2.1.2. Timeout

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

2.2.1.3. Objective

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

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

An essential requirement for all language mappings and all implementations is that, regardless of what other options exist for a language-specific represenation of the value, there is always an option to use a CBOR byte string as the value. The API will then wrap this byte string in CBOR Tag 24 for transmission via GRASP, and unwrap it after reception.

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

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

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

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

2.2.1.4. ASA_locator

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

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

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

2.2.1.5. Tagged_objective

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

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

2.2.1.6. Asa_nonce

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

2.2.1.7. Session_nonce

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

2.2.2. Registration

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

- register_asa()

  Input parameter:

  name of the ASA (UTF-8 string)

  Return parameters:

  errorcode (integer)

  asa_nonce (integer) (if successful)

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

- deregister_asa()

  Input parameters:

  asa_nonce (integer)

  name of the ASA (UTF-8 string)

  Return parameter:

  errorcode (integer)

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

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

- register_objective()

  Input parameters:

asa_nonce (integer)

objective (structure)

ttl (integer - default GRASP_DEF_TIMEOUT)

discoverable (Boolean - default False)

overlap (Boolean - default False)

local (Boolean - default False)

Return parameter:

errorcode (integer)

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

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

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

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

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

This call may be repeated for multiple objectives.

- deregister_objective()

  Input parameters:

asa_nonce (integer)

objective (structure)

Return parameter:

errorcode (integer)

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

2.2.3. Discovery

- discover()

Input parameters:

asa_nonce (integer)

objective (structure)

timeout (integer)

flush (Boolean - default False)

Return parameters:

errorcode (integer)

locator_list (structure)

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

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

2.2.4. Negotiation

- request_negotiate()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

  peer (ASA_locator)

  timeout (integer)

  Return parameters:

  errorcode (integer)

  session_nonce (structure) (if successful)

  proffered_objective (structure) (if successful)

  reason (string) (if negotiation declined)

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

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

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

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

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

  2. The 'session_nonce' parameter is not null. In this case negotiation must continue. The returned 'proffered_objective' contains the first value proffered by

the negotiation peer. Note that this instance of the objective must be used in the subsequent negotiation call because it also contains the current loop count. The 'session_nonce' must be presented in all subsequent negotiation steps.

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

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

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

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

- listen_negotiate()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

  Return parameters:

  errorcode (integer)

  session_nonce (structure) (if successful)

  requested_objective (structure) (if successful)

  This function instructs GRASP to listen for negotiation requests for the given 'objective'. It also enables discovery

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

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

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

- stop_listen_negotiate()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

  Return parameter:

  errorcode (integer)

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

- negotiate_step()

  Input parameters:

  asa_nonce (integer)

  session_nonce (structure)

objective (structure)

timeout (integer)

Return parameters:

Exactly as for ’request_negotiate’

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

- negotiate_wait()

  Input parameters:

  asa_nonce (integer)

  session_nonce (structure)

  timeout (integer)

  Return parameters:

  errorcode (integer)

  Delay negotiation session by ’timeout’ milliseconds.

- end_negotiate()

  Input parameters:

  asa_nonce (integer)

  session_nonce (structure)

  reply (Boolean)

  reason (UTF-8 string)

  Return parameters:

  errorcode (integer)

  End the negotiation session.

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

'reason' = optional string describing reason for decline.

2.2.5. Synchronization and Flooding

- synchronize()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

  peer (ASA_locator)

  timeout (integer)

  Return parameters:

  errorcode (integer)

  objective (structure) (if successful)

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

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

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

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

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

  Call in a separate thread if asynchronous operation is required.

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

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

- listen_synchronize()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

  Return parameters:

  errorcode (integer)

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

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

- stop_listen_synchronize()

  Input parameters:

  asa_nonce (integer)

  objective (structure)

  Return parameters:

  errorcode (integer)

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

- flood()

  Input parameters:

  asa_nonce (integer)

  ttl (integer)

  tagged_objective_list (structure)

Return parameters:

errorcode (integer)

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

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

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

The function checks that the ASA registered each objective.

This call may be repeated whenever any value changes.

- get_flood()

Input parameters:

asa_nonce (integer)

objective (structure)

Return parameters:

errorcode (integer)

tagged_objective_list (structure) (if successful)

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

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

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

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

- expire_flood()

  Input parameters:

  asa_nonce (integer)

  tagged_objective (structure)

  Return parameters:

  errorcode (integer)

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

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

2.2.6. Invalid Message Function

- send_invalid()

  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. Non-threaded Implementations

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

The calls to which this mechanism applies are:

discover()

request_negotiate()

negotiate_step()

listen_negotiate()

synchronize()

4. Example Logic Flows

TBD

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

5. Security Considerations

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

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

6. IANA Considerations

This does not need IANA assignment.

7. Acknowledgements

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

Excellent suggestions were made by Michael Richardson.

8. References

8.1. Normative References

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

8.2. Informative References

[I-D.ietf-anima-autonomic-control-plane]
Behringer, M., Eckert, T., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-ietf-anima-autonomic-control-plane-10 (work in progress), September 2017.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-07 (work in progress), July 2017.

[I-D.ietf-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., Pierre, P., Liu, B., Nobre, J., and J. Strassner, "A Reference Model for Autonomic Networking", draft-ietf-anima-reference-model-04 (work in progress), July 2017.

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

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

Appendix A. Error Codes

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

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

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

draft-liu-anima-grasp-api-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
Department of Computer Science
University of Auckland
PB 92019
Auckland 1142
New Zealand

Email: brian.e.carpenter@gmail.com

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

Email: leo.liubing@huawei.com

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

Email: wdwang@bupt.edu.cn

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

Email: xygong@bupt.edu.cn

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 27, 2017

B. Liu
S. Jiang
Huawei Technologies

May 26, 2017

Information Distribution over GRASP
draft-liu-anima-grasp-distribution-04

Abstract

This document discusses the requirement of information distribution capability in autonomic networks. Ideally, the autonomic network should support distributing some information which is generated/ injected at an arbitrary autonomic node and be distributed among the whole autonomic domain. This docuemnt specifically proposes to achive this goal based on the GRASP (A Generic Autonomic Signaling Protocol), and specifies additional node behavior.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 27, 2017.

Table of Contents

1. Introduction

In an autonomic network, sometimes the nodes need to share a set of common information. One typical case is the Intent Distribution which is briefly discussed in Section 4.5 of [I-D.behringer-anima-reference-model]. However, the distribution should be a general function that one autonomic node should support, rather than a specific mechanism dedicated for Intent. This document firstly analyzes several basic information distribution scenarios (Section 2), and then discusses the technical requirements (Section 3) that one autonomic node needs to fulfill.

This document proposes to achieve distribution function based on the GRASP (A Generic Autonomic Signaling Protocol) [I-D.ietf-anima-grasp]

. GRASP already provides some capability to support part of the distribution function. Along with that, this document also proposes some additional functionality. Detailed design is described in Section 4.

2. Information Distribution Scenarios

2.1. Whole Domain Distribution

Once the information is input to the autonomic network, the node that firstly handle the information MUST be able to distribute it to all the other nodes in the autonomic domain.

The distributed information might not relevant to every autonomic node, but it is flooded to all the devices.

2.2. Selective Distribution

When one node receive the information, it only replicates it to the neighbors that fit for a certain of conditions. This could reduce some unnecessary signaling amplification.

However, this scenario implies there needs to be corresponding mechanisms to represent the conditions and to judge which neighbors fit for the conditions. Please refer to Section 4.3.2 (selective flooding behavior).

2.3. Incremental Distribution

The distribution only goes to the nodes that newly get online. This might mostly happen between neighbors.

The incremental distribution could also be a sub scenario of the whole domain distribution. When one node is doing the whole domain distribution, it is possible that some of its neighbors are sleeping/ off-line, so when the neighbors get online again, the node should do incremental distribution of the previous whole domain distributed information.

3. Distribution Requirements

3.1. Identifying Autonomic Domain Boundary

The domain boundary devices are supposed to know themselves as boundary. When the distribution messages come to the devices, they do not distribute them outside the domain.

3.2. Arbitrary Injecting Point

The distributed information SHOULD be injected at any autonomic node within the domain (or within a specific set of nodes [TBD]).

3.3. Avoiding Loops

There should be a mechanism to prevent the distributed information to travel around the domain again and again, so that there would not be a large amount of redundant packets troubling the network.

3.4. Selective Flooding

When one node receive the information, it only floods it to the neighbors that fit for a certain of rules.

3.5. Point-to-Point Distribution

One node only distributes the information to another node. This is for the incremental distribution scenario.

3.6. Verification of Distributed Information

- Information integrity verification

  The receiving node SHOULD be able to verify whether the distributed information is from the certain node. In other words, it needs to make sure the information hasn’t been modified.

- Source authorization verification

  Even the information integrity was verified, the distributed information might still be invalid, since the distribution source might not have the right to distribute such information that it just exceeds its authority.

3.7. Conflict Handling

As long as it supports arbitrary point of injecting distribution, there is possibility that two nodes advertise the same information but with conflict attribute(s). Hence, there should be a mechanism to handle the conflict.

4. Distribution Function and Behavior Specification

This section specifies using certain GRASP messages for distribution, and also specifies the distribution behavior in an autonomic node.

4.1. Using GRASP Flood Synchronization Message

It is natural to use the GRASP Flood Synchronization message for distribution, since the Flood Synchronization behavior specified in GRASP is identical to the the whole domain distribution scenario described in Section 2.1. And the Flood Synchronization naturally fits for "Arbitrary Injection Point" and "Avoiding Loops" requirements.

4.2. Using GRASP Synchronization Message

It is natural to use the GRASP Synchronization message for Point-to-Point distribution. The two behavior is identical.

4.3. Selective Flooding

4.3.1. Selecting Cretiria

When doing selective flooding, the distributed information needs to contain the cretiria for nodes to judge which interfaces should be sent the distributed information and which are not. Specifically, the cretiria contains:

- Matching condition: a set of matching rules.
- Matching object: the object that the match condition would be applied to. For example, the matching object could be node itself or its neighbors.
- Action: what behavior the node needs to do when the matching object matches or failed the matching condition. For example, the action could be forwarding or discarding the distributed message.

Example:

- Matching condition: "Device role=IPRAN_RSG"
- Matching objective: "Neighbors"
- Action: "Forward"

This example means: only distributing the information to the neighbors who are IPRAN_RSG.

4.3.2. Node Behavior

1) The distribution initial node Includes the Selecting Cretiria information in the message that carries the distributed information.

2) The recieving node decides the action according to the Selecting Cretiria carried in the message.

2-1 When the Matching Object is "Neighbors", then the node matches the relevant information of its neighbors to the Matching Condition. If the node finds one neighbor matches the Matching Condition, then it forwards the distributed messge to the neighbor. If not, the node discards forwarding the message to the neighbor.

2-2 When the Matching Object is the node itself, then the node matches the relevant information of its own to the Matching Condition. If the node finds itself matches the Matching Condition, then it forwards the distributed messge to its neighbors; if not, the node discards forwarding the message to the neighbors.

4.4. Conflict Handling

The distribution information needs to include timestamps or version information. When conflict happens, the node only accepts the latest information.

4.5. Distribution Source Authentication

The distribution source authentication could be done at multiple layers:

- Outer layer authentication: the GRASP communication is within ACP (Autonomic Control Plane, [I-D.behringer-anima-autonomic-control-plane]). This is the default GRASP behavior.

- 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.

5. Security Considerations

TBD.

6. IANA Considerations

No IANA assignment is needed.

7. Acknowledgements

This document is inherited from [I-D.ietf-anima-grasp] and [I-D.behringer-anima-reference-model]. So thanks all the contributors of the two work items.

This document was produced using the xml2rfc tool [RFC2629].

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-12 (work in progress), May 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>.

[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>.

8.2. Informative References

[I-D.behringer-anima-autonomic-control-plane]
Behringer, M., Bjarnason, S., BL, B., and T. Eckert, "An Autonomic Control Plane", draft-behringer-anima-autonomic-control-plane-03 (work in progress), June 2015.

[I-D.behringer-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., Liu, B., Jeff, J., and J. Strassner, "A Reference Model for Autonomic Networking", draft-behringer-anima-reference-model-04 (work in progress), October 2015.

[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.irtf-nmrg-autonomic-network-definitions]
Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking - Definitions and Design Goals", draft-irtf-nmrg-autonomic-network-definitions-07 (work in progress), March 2015.

[I-D.pritikin-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., and S. Bjarnason, "Bootstrapping Key Infrastructures", draft-pritikin-anima-bootstrapping-keyinfra-02 (work in progress), July 2015.

Authors' Addresses

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

Email: leo.liubing@huawei.com

Sheng Jiang
Huawei Technologies
Q14, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: jiangsheng@huawei.com