Network Working Group   B. Carpenter
Internet-Draft   Univ. of Auckland
Intended status: Informational   L. Ciavaglia
Expires: July 11, 2019   Nokia
                                                               S. Jiang
                                              Huawei Technologies Co., Ltd
                                                              P. Peloso
                                                                   Nokia
                                                        January 7, 2019

Guidelines for Autonomic Service Agents
draft-carpenter-anima-asa-guidelines-06

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 July 11, 2019.

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."

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

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

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

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

2. Logical Structure of an Autonomic Service Agent

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

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

o Obtain authorization credentials.

o Register the ASA with GRASP.

o Acquire relevant policy parameters.

o Define data structures for relevant GRASP objectives.

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

o Launch a self-monitoring thread.

o 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:

o A background thread to repeatedly flood an objective to the AN, so
   that any ASA can receive the objective's latest value.

o A thread to accept incoming synchronization requests for an
   objective managed by this ASA.

o 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.

o 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.

Not all programming environments explicitly support multi-threading.
In such cases, 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 operations.

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 parameters that affect its operation. It
   should also do whatever is required to avoid unnecessary resource
   consumption, such as including an arbitrary wait time in each cycle
   of the main loop.

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

   Appendix B presents some example logic flows in informal pseudocode.

3. Interaction with the Autonomic Networking Infrastructure

3.1. Interaction with the security mechanisms

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

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

3.2. Interaction with the Autonomic Control Plane

   In a normal autonomic network, ASAs will run as 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.ietf-anima-grasp-api] available in user space.
   Thus ASAs may operate without special privilege, unless they need it
   for other reasons. The ASA's view of GRASP is built around GRASP
   objectives (Section 5), defined as data structures containing
   administrative information such as the objective's unique name, and
   its current value. The format and size of the value is not
   restricted by the protocol, except that it must be possible to
   serialise it for transmission in CBOR [RFC7049], which is no
   restriction at all in practice.

   The GRASP API should offer the following features:

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

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

   o 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.

   o 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.

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

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

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

3.4. Interaction with policy mechanism

   At the time of writing, the policy (or "Intent") mechanism for the
   ANI is undefined. 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 policy, so every
   ASA implementation must include carefully chosen default values and
   settings for all policy parameters.

4. Interaction with Non-Autonomic Components

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

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

5. Design of GRASP Objectives

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

   In particular, if 'dry run' mode is defined for the objective, its
   specification, and every implementation, must consider what state

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

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

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

6. Life Cycle

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

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

```
                          +--------------+
      Undeployed ------>  |              |  ------> Undeployed
                          |   Installed  |
                    +-->  |              |  ---+
      Mandate      |      +--------------+      |   Receives a
      is revoked   |      +--------------+      |    Mandate
                +---      |              |      <--+
                          | Instantiated |
                    +-->  |              |  ---+
         set       |      +--------------+      |   set
        down       |      +--------------+      |    up
                +---      |              |      <--+
                          |  Operational |
                          |              |
                          +--------------+
```
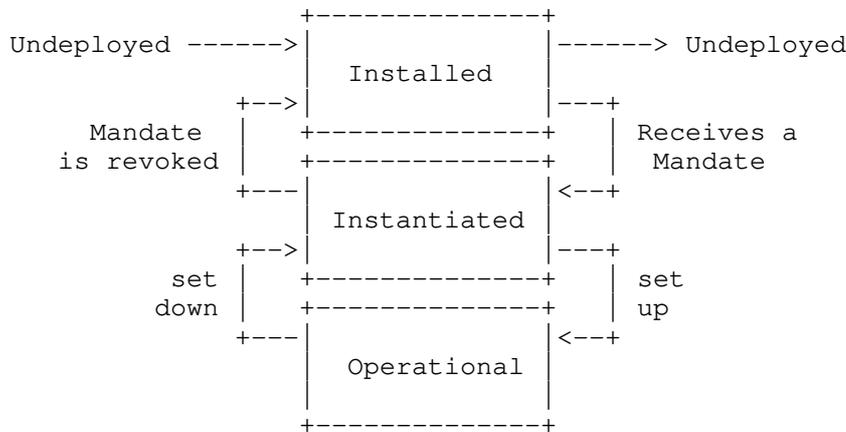
        Figure 1: Life cycle of an Autonomic Service Agent

6.1. Installation phase

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

   There are 3 properties applicable to the installation of ASAs:

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

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

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

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

6.1.1. Installation phase inputs and outputs

   Inputs are:

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

   [Installation_target_Infrastructure] that specifies the candidate
      Installation Hosts,

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

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

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

6.2. Instantiation phase

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

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

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

6.2.1. Operator's goal

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

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

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

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

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

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

6.2.2. Instantiation phase inputs and outputs

Inputs are:

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

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

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

Outputs are:

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

6.2.3. Instantiation phase requirements

   The instructions described in section 4.2 could be either:

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

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

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

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

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

   o the coordination entities (see [I-D.ciavaglia-anima-coordination]
      - Autonomic Functions Coordination)

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

6.3. Operation phase

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

   During the Operation phase, the operator can:

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

Modify ASAs targets: meaning setting them different objectives.

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

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

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

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

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

then execute their control loop under coordination supervision/ instructions

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

7. Coordination between Autonomic Functions

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

8. Coordination with Traditional Management Functions

Some ASAs will have functions that overlap with existing configuration tools and network management mechanisms such as command line interfaces, DHCP, DHCPv6, SNMP, NETCONF, RESTCONF and YANG-based solutions. Each ASA designer will need to consider this issue and how to avoid clashes and inconsistencies. Some specific

considerations for interaction with OAM tools are given in
[I-D.ietf-anima-stable-connectivity]. As another example,
[I-D.ietf-anima-prefix-management] describes how autonomic management
of IPv6 prefixes can interact with prefix delegation via DHCPv6. The
description of a GRASP objective and of an ASA using it should
include a discussion of any such interactions.

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

9. Robustness

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

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

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

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

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

   5. 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).

10. Security Considerations

   ASAs are intended to run in an environment that is protected by the
   Autonomic Control Plane [I-D.ietf-anima-autonomic-control-plane],
   admission to which depends on an initial secure bootstrap process
   [I-D.ietf-anima-bootstrapping-keyinfra]. 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.

11. IANA Considerations

   This document makes no request of the IANA.

12. Acknowledgements

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

13. References

13.1. Normative References

   [I-D.ietf-anima-autonomic-control-plane]
              Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic
              Control Plane (ACP)", draft-ietf-anima-autonomic-control-
              plane-18 (work in progress), August 2018.

[I-D.ietf-anima-bootstrapping-keyinfra]
          Pritikin, M., Richardson, M., Behringer, M., Bjarnason,
          S., and K. Watsen, "Bootstrapping Remote Secure Key
          Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-
          keyinfra-17 (work in progress), November 2018.

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

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

13.2. Informative References

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

[GANA13]   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-grasp-api]
          Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic
          Autonomic Signaling Protocol Application Program Interface
          (GRASP API)", draft-ietf-anima-grasp-api-02 (work in
          progress), June 2018.

   [I-D.ietf-anima-prefix-management]
             Jiang, S., Du, Z., Carpenter, B., and Q. Sun, "Autonomic
             IPv6 Edge Prefix Management in Large-scale Networks",
             draft-ietf-anima-prefix-management-07 (work in progress),
             December 2017.

   [I-D.ietf-anima-reference-model]
             Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L.,
             and J. Nobre, "A Reference Model for Autonomic
             Networking", draft-ietf-anima-reference-model-10 (work in
             progress), November 2018.

   [I-D.ietf-anima-stable-connectivity]
             Eckert, T. and M. Behringer, "Using Autonomic Control
             Plane for Stable Connectivity of Network OAM", draft-ietf-
             anima-stable-connectivity-10 (work in progress), February
             2018.

   [I-D.ietf-core-yang-cbor]
             Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A.
             Minaburo, "CBOR Encoding of Data Modeled with YANG",
             draft-ietf-core-yang-cbor-07 (work in progress), September
             2018.

   [I-D.irtf-nfvrg-gaps-network-virtualization]
             Bernardos, C., Rahman, A., Zuniga, J., Contreras, L.,
             Aranda, P., and P. Lynch, "Network Virtualization Research
             Challenges", draft-irtf-nfvrg-gaps-network-
             virtualization-10 (work in progress), September 2018.

   [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>.

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

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

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

   Expanded and improved example logic flow.

   Editorial corrections.

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

   Added section on relationshp with non-autonomic components.

   Editorial corrections.

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

   Added note about simple ASAs.

   Added note about NFV/SFC services.

   Improved text about threading v event loop model

   Added section about coordination with traditional tools.

   Added appendix with example logic flow.

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

   Added details on life cycle.

   Added details on robustness.

   Added co-authors.

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

   Expanded description of event-loop case.

Added note about 'dry run' mode.

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

More sections filled in

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

Initial version

Appendix B. Example Logic Flows

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

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

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

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

When running as a delegator, it starts with an empty resource pool,
it acquires the policy parameters by GRASP synchronization, and it
delegates quantities of the resource to consumers that request it.
Both as a master and as a delegator, when its pool is low it seeks
quantities of the resource by requesting GRASP negotiation with peer

ASAs. When its pool is sufficient, it hands out resource to peer
ASAs in response to negotiation requests. Thus, over time, the
initial resource pool held by the master will be shared among all the
delegators according to demand.

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

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

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

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

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

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

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

The GRASP objectives are as follows:

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

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

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

```
MAIN PROGRAM:

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

MAIN_NEGOTIATOR thread:

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

```
   NEGOTIATOR thread:

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

   DELEGATOR thread:

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

   SYNCHRONIZER thread:

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

   FLOODER thread:

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

```
GARBAGE_COLLECTOR thread:

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

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

       Transferring Bulk Data over the GeneRic Autonomic Signaling Protocol
                                  (GRASP)
                    draft-carpenter-anima-grasp-bulk-03

Abstract

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

Status of This Memo

Copyright Notice

the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

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. A related case
   might be installing the code of a new or updated ASA to a target node
   (see the discussion of ASA life cycles in
   [I-D.carpenter-anima-asa-guidelines]).

   Naturally, an existing solution such as a secure file transfer
   protocol or secure HTTP might be used for this. Other management
   protocols such as syslog [RFC5424] or NETCONF [RFC6241] might also be

used for related purposes, or might be mapped directly over GRASP.
The present document, however, applies to any scenario where it is
preferable to re-use the autonomic networking infrastructure itself
to transfer a significant amount of data, rather than install and
configure an additional mechanism. The basic model is to use the
GRASP Negotiation process to transfer and acknowledge multiple blocks
of data in successive negotiation steps.

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

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. No changes or extensions are
required to GRASP itself, but compared to a normal GRASP negotiation,
the communication pattern is slightly asymmetric:

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

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

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

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

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

Errors of any kind are handled with the normal GRASP mechanisms, in particular by an M_END message with an O_DECLINE option in either direction.

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

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

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

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

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

3. Example for File Transfer

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

Firstly we define a GRASP objective informally:

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

The formal CDDL definition [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:

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

   o 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.

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

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

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

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

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

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

#file received
exit
```

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

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

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

4. Loss Detection

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

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

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

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

5. Maximum Transmission Unit

   In an IPv6 environment, a minimal MTU of 1280 bytes can be assumed,
   and assuming that high throughput is not a requirement, bulk
   transfers can be designed to match that MTU. However, there are
   environments where the underlying physical MTU is much smaller. For
   example, on an IEEE 802.15.4 network it may be less than 100 bytes
   [RFC4944]. 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. Indeed, the presence of such
       an adaptation layer may be impossible to detect.

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

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

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

6. Pipelining

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

7. Other Considerations

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

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

8. Possible Future Work

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

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

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

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

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

9. Implementation Status [RFC Editor: please remove]

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

10. Security Considerations

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

11. IANA Considerations

   This document makes no request of the IANA.

12. Acknowledgements

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

13. References

13.1. Normative References

   [I-D.ietf-anima-grasp]
              Bormann, C., Carpenter, B., and B. Liu, "A Generic
              Autonomic Signaling Protocol (GRASP)", 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 and JSON data structures", draft-ietf-cbor-
              cddl-06 (work in progress), November 2018.

13.2. Informative References

   [I-D.carpenter-anima-asa-guidelines]
              Carpenter, B., Ciavaglia, L., Jiang, S., and P. Pierre,
              "Guidelines for Autonomic Service Agents", draft-
              carpenter-anima-asa-guidelines-05 (work in progress), June
              2018.

   [I-D.ietf-anima-grasp-api]
              Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic
              Autonomic Signaling Protocol Application Program Interface
              (GRASP API)", draft-ietf-anima-grasp-api-02 (work in
              progress), June 2018.

   [I-D.ietf-anima-reference-model]
              Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L.,
              and J. Nobre, "A Reference Model for Autonomic
              Networking", draft-ietf-anima-reference-model-10 (work in
              progress), November 2018.

   [I-D.liu-anima-grasp-distribution]
              Liu, B., Jiang, S., Xiao, X., Hecker, A., and Z.
              Despotovic, "Information Distribution in Autonomic
              Networking", draft-liu-anima-grasp-distribution-09 (work
              in progress), October 2018.

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

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

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

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

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

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

   Added future work section, implementation status.

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

   Update reference, fix TBDs.

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

   Updates after IETF100 discussion.

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

   Initial version.

Authors' Addresses

   Brian Carpenter
   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

                       An Autonomic Control Plane (ACP)
                  draft-ietf-anima-autonomic-control-plane-18

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 Operations Administration and Management
   (OAM) communications over a network that is secure and reliable even
   when the network is not configured, or misconfigured.

Table of Contents

1. Introduction (Informative)

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

   Autonomic functions need an autonomically built communications
   infrastructure. This infrastructure needs to be secure, resilient
   and re-usable by all autonomic functions. Section 5 of [RFC7575]

introduces that infrastructure and calls it the Autonomic Control
Plane (ACP). More descriptively it would be the "Autonomic
communications infrastructure for 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 uses a
routing and forwarding table which is dependent on correct
configuration and routing. Misconfigurations or routing problems can
disrupt management and control channels. Traditionally, an out-of-
band network has been used to avoid or allow recovery from such
problems, or personnel are sent on site to access devices through
out-of-band management ports (also called craft ports, serial
console, management ethernet port). However, both options are
expensive.

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

This document describes a modular design for a self-forming, self-
managing and self-protecting 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 is achieved are described in Section 6. The ACP is designed to
remain 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-Plane 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:

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

2. A controller or network management system can use it to securely
    bootstrap network devices in remote locations, even if the (Data-
    Plane) network in between is not yet configured; no Data-Plane
    dependent bootstrap configuration is required. An example of
    such a secure bootstrap process is described in
    [I-D.ietf-anima-bootstrapping-keyinfra].

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

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

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

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

The ACP provides secure IPv6 connectivity, therefore it cannot only
be used as the secure connectivity for self-management as required
for the ACP in [RFC7575], but it can also be used as the secure
connectivity for traditional (centralized) management. The ACP can
be implemented and operated without any other components of autonomic
networks, except for the GRASP protocol which it leverages.

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

Combining ACP with Bootstrapping Remote Secure Key Infrastructures
(BRSKI), see [I-D.ietf-anima-bootstrapping-keyinfra]) results in the
"Autonomic Network Infrastructure" as defined in
[I-D.ietf-anima-reference-model], which provides autonomic

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

1.1. Applicability and Scope

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

The design of the ACP as defined in this document is considered to be
applicable to all types of "professionally managed" networks: Service
Provider, Local Area Network (LAN), Metro(politan networks), Wide
Area Network (WAN), Enterprise Information Technology (IT) and
->"Operational Technology" () (OT) networks. The ACP can operate
equally on layer 3 equipment and on layer 2 equipment such a bridges
(see Section 7). The encryption mechanism used by the ACP is defined
to be negotiable, therefore it can be extended to environments with
different encryption protocol preferences. The minimum
implementation requirements in this document attempt to achieve
maximum interoperability by requiring support for few options: IP
security (IPsec), see [RFC4301]) and datagram Transport Layer
Security version 1.2 (DTLS), see [RFC6347]), depending on type of
device.

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

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

The protocol choices of the ACP are primarily based on wide use and
support in networks and devices, well understood security properties
and required scalability. The ACP design is an attempt to produce

the lowest risk combination of existing technologies and protocols to
build a widely applicable operational network management solution:

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

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

The ACP design can be applicable to (cpu, memory) constrained devices
and (bitrate, reliability) constrained networks, but this document
does not attempt to define the most constrained type of devices or
networks to which the ACP is applicable. RPL and DTLS are two
protocol choices already making ACP more applicable to constrained
environments. See Appendix A.9 for discussions about how future
standards or proprietary extensions/variations of the ACP could
better meet different expectations from those on which the current
design is based.

2. Acronyms and Terminology (Informative)

[RFC Editor: WG/IETF/IESG review of the terms below asked for
references between these terms when they refer to each other. The
only option I could fin RFC/XML to point to a hanging text acronym
definition that also displays the actual term is the format="title"
version, which leads to references such as '->"ACP domain
certificate" ()'. I found no reasonable way to eliminate the
trailing '()' generated by this type of cross references. Can you
please take care of removing these artefacts during editing (after
conversion to nroff ?). I also created a ticket to ask for an
xml2rfc enhancement to avoid this in the future:
https://trac.tools.ietf.org/tools/xml2rfc/trac/ticket/347.

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

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

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

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

ACP address: An IPv6 address assigned to the ACP node. It is stored in the 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 interface: An interface on an ACP node providing access to the ACP for non ACP capable nodes without using an ACP secure channel. See Section 8.1.1.

ACP domain: The ACP domain is the set of nodes with ->"ACP domain certificates" that allow them to authenticate each other as members of the ACP domain. See also Section 6.1.2.

ACP (ANI/AN) Domain Certificate: A provisioned [RFC5280] 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.

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 Loopback interface in the ACP Virtual Routing and Forwarding (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 /48 IPv6 address prefixes used 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 cryptographically authenticated and encrypted
   data connection established between (normally) adjacent ACP nodes
   to carry traffic of the ACP VRF secure and isolated from Data-
   Plane traffic in-band over the same link/path as the Data-Plane.

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

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

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 (automated)
   bootstrap and stable connectivity for SDN networks - see
   [RFC8368].

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
     routing and forwarding in the node other than the ACP VRF. In a
     simple ACP or ANI node, the Data-Plane is typically provisioned by
     means other than autonomically, for example manually (including
     across the ACP) or via SDN controllers. In a fully 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!

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". This instance of GRASP runs across
     the ACP secure channels to support BRSKI and other NOC/OAM or
     Autonomic Functions. See [I-D.ietf-anima-grasp].

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

in-band (management): The type of management used predominantly in
     IP based networks, not leveraging an ->"out-of-band network" ().
     In in-band management, access to the managed equipment depends on

the configuration of this equipment itself: interface, addressing, forwarding, routing, policy, security, management. This dependency makes in-band management fragile because the configuration actions performed may break in-band management connectivity. Breakage can not only be unintentional, it can simply be an unavoidable side effect of being unable to create configuration schemes where in-band management connectivity configuration is unaffected by Data-Plane configuration. See also ->"(virtual) out-of-band network" ().

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

Loopback interface: The conventional name for an internal IP interface to which addresses may be assigned, but which transmits no external traffic.

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.

Node-ID: The identifier of an ACP node inside that ACP. It is the last 64 (see Section 6.10.3) or 78-bits (see Section 6.10.5) of the ACP address.

Operational Technology (OT): "https://en.wikipedia.org/wiki/ Operational_Technology" [1]: "The hardware and software dedicated to detecting or causing changes in physical processes through direct monitoring and/or control of physical devices such as valves, pumps, etc.". OT networks are today in most cases well separated from Information Technology (IT) networks.

(virtual) out-of-band network: An out-of-band network is a secondary network used to manage a primary network. The equipment of the primary network is connected to the out-of-band network via dedicated management ports on the primary network equipment. Serial (console) management ports were historically most common,

higher end network equipment now also has ethernet ports dedicated
only for management. An out-of-band network provides management
access to the primary network independent of the configuration
state of the primary network. One of the goals of the ACP is to
provide this benefit of out-of-band networks virtually on the
primary network equipment. The ACP VRF acts as a virtual out of
band network device providing configuration independent management
access. The ACP secure channels are the virtual links of the ACP
virtual out-of-band network, meant to be operating independent of
the configuration of the primary network. See also ->"in-band
(management)" ().

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

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

(ACP/ANI/BRSKI) Registrar: An ACP registrar is an entity (software
   and/or person) that is orchestrating the enrollment of ACP nodes
   with the ACP domain certificate. ANI nodes use BRSKI, so ANI
   registrars are also called BRSKI registrars. For non-ANI ACP
   nodes, the registrar mechanisms are undefined by this document.
   See Section 6.10.7. Renewal and other maintenance (such as
   revocation) of ACP domain certificates may be performed by other
   entities than registrars. EST must be supported for ACP domain
   certificate renewal (see Section 6.1.3). BRSKI is an extension of
   EST, so ANI/BRSKI registrars can easily support ACP domain
   certificate renewal in addition to initial enrollment.

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: (Global ID prefix) 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]). The ULA Global ID prefix are the first 48-bits of a
   ULA address. In this document it is abbreviated as "ULA prefix".

(ACP) VRF: The ACP is modeled in this document as a "Virtual Routing
   and Forwarding" instance (VRF). This means that it is based on a
   "virtual router" consisting of a separate IPv6 forwarding table to

   which the ACP virtual interfaces are attached and an associated
   IPv6 routing table separate from the Data-Plane. Unlike the VRFs
   on MPLS/VPN-PE ([RFC4364]) or LISP XTR ([RFC6830]), the ACP VRF
   does not have any special "core facing" functionality or routing/
   mapping protocols shared across multiple VRFs. In vendor products
   a VRF such as the ACP-VRF may also be referred to as a so called
   VRF-lite.

   (ACP) Zone: An ACP zone is a set of ACP nodes using the same zone
      field value in their ACP address according to Section 6.10.3.
      Zones are a mechanism to support structured addressing of ACP
      addresses within the same /48-bit ULA prefix.

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

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

3.1. An Infrastructure for Autonomic Functions

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

3.2. Secure Bootstrap over a not configured Network

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

   With the ACP, secure bootstrap of new devices and whole new networks
   can happen without requiring any configuration of unconfigured
   devices along the path: As long as all devices along the path support
   ACP and a zero-touch bootstrap mechanism such as BRSKI, the ACP
   across a whole network of unconfigured devices can be brought up
   without operator/provisioning intervention. The ACP also provides

additional security for any bootstrap mechanism, because it encrypts
the traffic along the path hop-by-hop.

3.3. Data-Plane Independent Permanent Reachability

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

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

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

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

Assuming appropriate implementation (see Section 6.12.2 for more
details), the ACP provides reachability that is independent of the
Data-Plane. This allows the control plane and management plane to
operate more robustly:

o For management plane protocols, the ACP provides the functionality
   of a Virtual out-of-band (VooB) channel, by providing connectivity
   to all nodes regardless of their Data-Plane configuration, routing
   and forwarding tables.

o For control plane protocols, the ACP allows their operation even
   when the Data-Plane is temporarily faulty, or during transitional

events, such as routing changes, which may affect the control
plane at least temporarily. This is specifically important for
autonomic service agents, which could affect Data-Plane
connectivity.

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

4. Requirements (Informative)

The following requirements were identified as the basis for the
design of the ACP based on the above use-cases (Section 3). These
requirements are informative for this specification because they
(merely) represent the use-case requirements. The keywords are
highlighted ("_") to be different from RFC2119. The ACP as specified
in the normative parts of this document is meeting or exceeding these
use-case requirements:

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

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

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

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

ACP5:   The ACP _MUST_ provide security: Messages coming through the
        ACP MUST be authenticated to be from a trusted node, and
        SHOULD (very strong SHOULD) be encrypted.

Explanation for ACP4: In a fully autonomic network (AN), newly
written ASA could potentially all communicate exclusively via GRASP
with each other, and if that was assumed to be the only requirement

against the ACP, it would not need to provide IPv6 layer connectivity
between nodes, but only GRASP connectivity. Nevertheless, because
ACP also intends to support non-AN networks, it is crucial to support
IPv6 layer connectivity across the ACP to support any transport and
application layer protocols.

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

5. Overview (Informative)

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

   1. An ACP node creates a Virtual Routing and Forwarding (VRF)
       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. For each node in the candidate peer list, it then establishes a
       secure tunnel of the negotiated type. The resulting tunnels are
       then placed into the previously set up VRF. This creates an
       overlay network with hop-by-hop tunnels.

   5. Inside the ACP VRF, each node assigns its ULA IPv6 address to a
       Loopback interface assigned to the ACP VRF.

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

   Note:

   o Non-autonomic NMS ("Network Management Systems") or SDN
      controllers have to be explicitly configured for connection into
      the ACP.

   o Connecting over non-ACP Layer-3 clouds requires explicit
      configuration. See Section 8.2.

   o None of the above operations (except explicit configured ones) are
      reflected in the configuration of the node.

   The following figure illustrates the ACP.

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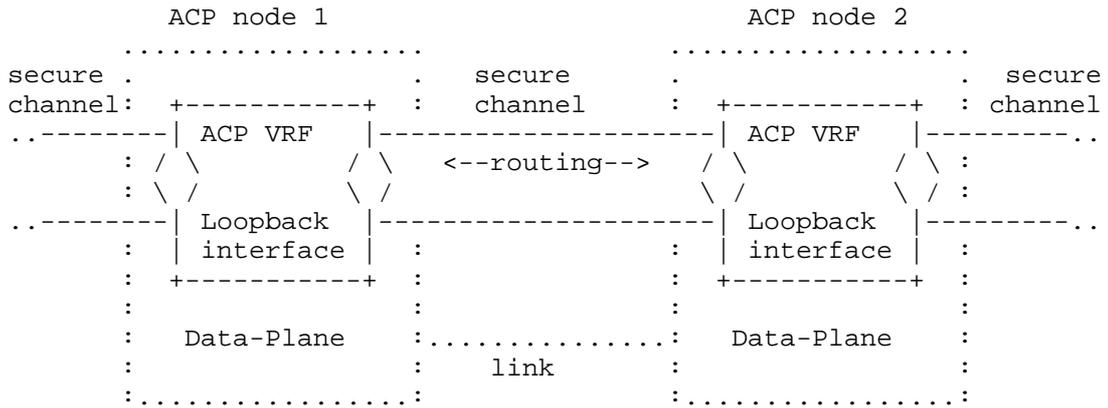
                 Figure 1: ACP VRF and secure channels

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

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

   This section 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 it's ACP domain
   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

   The 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, each ACP member requires keying material: An ACP node MUST
   have a certificate (LDevID) and a Trust Anchor (TA) consisting of a
   certificate (chain) used to sign the LDevID of all ACP domain
   members. The LDevID is used to cryptographically authenticate the
   membership of its owner node in the ACP domain to other ACP domain
   members, the TA is used to authenticate the ACP domain membership of
   other nodes (see Section 6.1.2).

   The LDevID is called the ACP domain certificate, the TA is the
   Certificate Authority (CA) 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
   Domain information field as specified in Section 6.1.1 in its domain
   certificate as well as those of candidate ACP peers. See
   Appendix A.2 for more information about enrollment or provisioning
   options.

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

   [RFC7575] defines the term "Autonomic Domain" as a collection of
   autonomic nodes. ACP nodes do not need to be fully autonomic, but
   when they are, then the ACP domain is an autonomic domain. Likewise,
   [I-D.ietf-anima-reference-model] defines the term "Domain
   Certificate" as the certificate used in an autonomic domain. The ACP
   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 SHOULD be used for any authentication
   between nodes with ACP domain certificates (ACP nodes and NOC nodes)
   where the required condition is ACP domain membership, such as ACP
   node to NOC/OAM end-to-end security and ASA to ASA end-to-end
   security. Section 6.1.2 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).

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 occurrences of rfcSELF in
this document with the RFC number assigned to this document and
remove this comment line]

```
domain-information = local-part "@" acp-domain-name
local-part = key [ "." local-info ]
key = "rfcSELF"
local-info = [ acp-address ] [ "+" rsub extensions ]
acp-address = 32hex-dig | 0
hex-dig = DIGIT / "a" / "b" / "c" / "d" / "e" / "f"
rsub = [ <subdomain> ] ; <subdomain> as of RFC1034, section 3.5
routing-subdomain = [ rsub " ." ] acp-domain-name
acp-domain-name = ; <domain> ; as of RFC 1034, section 3.5
extensions = *( "+" extension )
extension = ; future standard definition.
            ; Must fit RFC5322 simple dot-atom format.

Example:
domain-information = rfcSELF+fd89b714f3db00000200000064000000
                     +area51.research@acp.example.com
acp-domain-name    = acp.example.com
routing-subdomain  = area51.research.acp.example.com
```

Figure 2: ACP Domain Information Field ABNF

Nodes complying with this specification MUST be able to receive their
ACP address through the domain certificate, in which case their own
ACP domain certificate MUST have the 32hex-dig "acp-address" field.
Nodes complying with this specification MUST also be able to
authenticate nodes as ACP domain members / ACP secure channel peers
when they have an empty or 0-value acp-address field. See
Section 6.1.2.

"acp-domain-name" is used to indicate the ACP Domain across which all
ACP nodes trust each other and are willing to build ACP channels to
each other. See Section 6.1.2. Acp-domain-name SHOULD be the FQDN
of a DNS 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 it intends to be equally
unique.

"routing-subdomain" is the autonomic subdomain composed of "rsub" and
"acp-domain-name". "rsub" is optional. When not present, "routing-
subdomain" is the same as "acp-domain-name". "routing-subdomain"
determines the /48 ULA prefix for ACP addresses. "rsub" therefore
allows to use multiple /48 ULA prefixes in an ACP domain. See
Appendix A.7 for example use-cases.

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

Formatting notes:

o "rsub" needs to be in the "local-part": If the format just had
   routing-subdomain as the domain part of the domain-information,
   rsub and acp-domain-name could not be separated from each other.
   It also makes acp-domain-name a valid e-mail target across all
   routing-subdomains.

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

o If "acp-address" is empty, and "rsub" is empty too, the "local-
   part" will have the format "rfcSELF + + extension(s)". The two
   plus characters are necessary so the node can unambiguously parse
   that both "acp-address" and "rsub" are empty.

o 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:

o It should be possible to share the LDevID with other uses beside
   the ACP. Therefore, the information element required for the ACP
   should be encoded so that it minimizes the possibility of creating
   incompatibilities with such other uses.

o The information for the ACP should not cause incompatibilities
   with any pre-existing ASN.1 software. This eliminates the
   introduction of a novel information element because that could
   require extensions to such pre-existing ASN.1 parsers.

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

o The element required for the ACP should not be misinterpreted by any other uses of the LDevID. If the element used for the ACP is interpreted by other uses, the impact should be benign.

o Using an IP address format encoding could result in non-benign misinterpretation of the domain information field; other uses 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 which does not belong to the ACP VRF.

o 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.

o rfc822Name encoding is quite flexible. The ACP information field encodes the full ACP address AND the domain name with rsub part, so that it is easier to examine/use the "domain information field".

o 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.

o 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 of a candidate peer certificate, independent of the protocol used:

1:    The peer certificate is valid (lifetime).

2:    The peer has proved ownership of the private key associated with the certificate's public key.

3:    The peer's certificate is signed by one of the trust anchors associated with the ACP domain certificate.

4:    If the node certificate indicates a Certificate Revocation List (CRL) Distribution Point (CDP) ([RFC5280], section 4.2.1.13) or Online Certificate Status Protocol (OCSP) responder ([RFC5280], section 4.2.2.1), then the peer's certificate must be valid according to those criteria: An OCSP check for the peer's certificate across the ACP must succeed or the peer certificate must not be listed in the CRL retrieved from the CDP.

5:    The peer's certificate has a syntactically valid ACP domain information field (encoded as subjectAltName / rfc822Name) and the acp-domain-name in that peer's domain information field is the same as in this ACP node's certificate.

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

6:    The candidate peer certificate's ACP domain information field has a non-empty acp-address field (either 32hex-dig or 0, according to Figure 2).

Rule 6: for the establishment of ACP secure channels ensures that they will only be built between nodes which indicate through the acp-address in their ACP domain certificate the ability and permission by the Registrar to participate in ACP secure-channels.

Nodes with an empty acp-address field can only use their ACP domain certificate for non-ACP-secure channel authentication purposes.

The special value 0 in an ACP certificates acp-address field is used for nodes that can and should determine their ACP address through other mechanisms than learning it through their ACP domain certificate. These ACP nodes are permitted to establish ACP secure channels. Mechanisms for those nodes to determine their ACP address are outside the scope of this specification.

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.

ACP nodes SHOULD be able to remember the EST server from which they
last renewed their ACP domain certificate and SHOULD provide the
ability for this remembered EST server to also be set by the ACP
Registrar (see Section 6.10.7) that initially enrolled the ACP device
with its ACP domain certificate. When BRSKI (see
[I-D.ietf-anima-bootstrapping-keyinfra]) is used, the ACP address of
the BRSKI registrar from the BRSKI TLS connection SHOULD be
remembered and used for the next renewal via EST if that registrar
also announces itself as an EST server via GRASP (see next section)
on its ACP address.

6.1.3.1. GRASP objective for EST server

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

        Example:

        [M_FLOOD, 12340815, h'fd89b714f3db0000200000064000001', 210000,
            ["SRV.est", 4, 255],
            [O_IPv6_LOCATOR,
                h'fd89b714f3db0000200000064000001', TCP, 80]
        ]

                        Figure 3: GRASP SRV.est example

The formal definition of the objective in Concise data definition
language (CDDL) (see [I-D.ietf-cbor-cddl]) is as follows:

    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 = any   ; Not used (yet)

                        Figure 4: GRASP SRV.est definition

The objective value "SRV.est" indicates that the objective is an
[RFC7030] compliant EST server because "est" is an [RFC6335]
registered service name for [RFC7030]. Objective-value MUST be
ignored if present. Backward compatible extensions to [RFC7030] MAY

be indicated through objective-value. Non [RFC7030] compatible
certificate renewal options MUST use a different objective-name.

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

6.1.3.2. Renewal

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

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

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

6.1.3.3. Certificate Revocation Lists (CRLs)

The ACP node SHOULD support Certificate Revocation Lists (CRL) via
HTTPs from one or more CRL Distribution Points (CDPs). The CDP(s)
MUST be indicated in the Domain Certificate when used. If the CDP
URL uses an IPv6 address (ULA address when using the addressing rules
specified in this document), the ACP node will connect to the CDP via
the ACP. If the CDP URL uses an IPv6 address (ULA address when using
the addressing rules specified in this document), the ACP node will

connect to the CDP via the ACP. If the CDP uses a domain name, the ACP node will connect to the CDP via the Data-Plane.

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

A CDP can be reachable across the ACP either by running it on a node with ACP or by connecting its node via an ACP connect interface (see Section 8.1). The CDP SHOULD use an ACP domain certificate for its HTTPs connections. The connecting ACP node SHOULD verify that the CDP certificate used during the HTTPs connection has the same ACP address as indicated in the CDP URL of the nodes ACP domain certificate

6.1.3.4. Lifetimes

Certificate lifetime may be set to shorter lifetimes than customary (1 year) because certificate renewal is fully automated via ACP and EST. The primary limiting factor for shorter certificate lifetimes is load on the EST server(s) and CA. It is therefore recommended that ACP domain certificates are managed via a CA chain where the assigning CA has enough performance to manage short lived certificates. See also Section 10.2.4 for discussion about an example setup achieving this.

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

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

6.1.3.5. Re-enrollment

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

In this role, the node does maintain the trust anchor and certificate chain associated with its ACP domain certificate exclusively for the

purpose of re-enrollment, and attempts (or waits) to get re-enrolled
with a new ACP certificate. The details depend on the mechanisms/
protocols used by the ACP registrars.

Please refer to Section 6.10.7 for explanations about ACP registrars
and vouchers as used in the following text.

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

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

Both when the BRSKI connection is attempted with the old ACP domain
certificate or the IDevID, the re-enrolling candidate ACP node SHOULD
authenticate the BRSKI registrar during TLS connection setup based on
its existing trust anchor/certificate chain information associated
with its old ACP certificate. The re-enrolling candidate ACP node
SHOULD only request a voucher from the BRSKI registrar when this
authentication fails during TLS connection setup.

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

Maintaining existing trust anchor information is especially important
when enrollment mechanisms are used that unlike BRSKI do not leverage
a voucher mechanism to authenticate the ACP registrar and where
therefore the injection of certificate failures could otherwise make
the ACP node easily attackable remotely.

When using BRSKI or other protocol/mechanisms supporting vouchers,
maintaining existing trust anchor information allows for re-
enrollment of expired ACP certificates to be more lightweight,

especially in environments where repeated acquisition of vouchers
during the lifetime of ACP nodes may be operationally expensive or
otherwise undesirable.

6.1.3.6. Failing Certificates

An ACP domain certificate is called failing in this document, if/when
the ACP node can determine that it was revoked (or explicitly not
renewed), or in the absence of such explicit local diagnostics, when
the ACP node fails to connect to other ACP nodes in the same ACP
domain using its ACP certificate. For connection failures to
determine the ACP domain certificate as the culprit, the peer should
pass the domain membership check (Section 6.1.2) and other reasons
for the connection failure can be excluded because of the connection
error diagnostics.

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

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

ACP nodes SHOULD support the option to determines whether its ACP
certificate is failing, and when it does, put itself into the role of
a re-enrolling candidate ACP node as explained above
(Section 6.1.3.5).

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
(identifier of the node inside the ACP, see Section 6.10.3 and
Section 6.10.5), interface on which neighbor was discovered (by GRASP
as explained below), link-local IPv6 address of neighbor on that
interface, certificate (including domain information field). 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 (i.e., not on a link
connected to this node), the information in the adjacency table can

be supplemented by configuration. For example, the Node-ID and IP
address could be configured.

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.

6.3. Neighbor Discovery with DULL GRASP

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

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

Note that the use of the IPv6 link-local multicast address
(ALL_GRASP_NEIGHBORS) implies the need to use Multicast Listener
Discovery Version 2 (MLDv2, see [RFC3810]) to announce the desire to
receive packets for that address. Otherwise DULL GRASP could fail to
operate correctly in the presence of MLD snooping, 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 on how to enable/disable ACP at node and
interface level. See Section 8.2.2 for more details about tunnels
(typical non-native interfaces). See Section 7 for how ACP should be
extended on devices operating (also) as L2 bridges.

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

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

```
[M_FLOOD, 12340815, h'fe80000000000000c0011001FEEF0000, 210000,
    ["AN_ACP", 4, 1, "IKEv2" ],
    [O_IPv6_LOCATOR,
        h'fe80000000000000c0011001FEEF0000, UDP, 15000]
    ["AN_ACP", 4, 1, "DTLS" ],
    [O_IPv6_LOCATOR,
        h'fe80000000000000c0011001FEEF0000, UDP, 17000]
]
```

Figure 5: GRASP AN_ACP example

The formal CDDL definition is:

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

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

objective-flags = sync-only ; as in the GRASP specification
sync-only =  4     ; M_FLOOD only requires synchronization
loop-count = 1     ; limit to link-local operation
objective-value = method
method = "IKEv2" / "DTLS"  ; or future standard methods
```

                   Figure 6: GRASP AN_ACP definition

   The objective-flags field is set to indicate synchronization.

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

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

   The session-id is a random number used for loop prevention
   (distinguishing a message from a prior instance of the same message).
   In DULL this field is irrelevant but 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 a string indicating the secure
   channel protocol available at the specified or implied locator.

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

   "IKEv2" is the 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). We therefore use the
   term "IKEv2" and not "IPsec" in the GRASP definitions 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).

"DTLS" indicates datagram Transport Layer Security version 1.2.
There is no default UDP port, it must always be locally assigned by
the node. See Section 6.7.2.

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

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

A node that supports more than one secure channel protocol method
needs to flood multiple versions of the "AN_ACP" objective so that
each method can be accompanied by its own locator-option. This can
use a single GRASP M_FLOOD message as shown in Figure 5.

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 established exclusively between nodes in the same domain.
This includes all routing subdomains. Appendix A.7 explains how ACP
connections across multiple routing subdomains are special.

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

6.5. 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 because that code exists already on
   them for end-to-end security, but low-end in-ceiling L2 switches may
   only want to support Media Access Control Security (MacSec, see
   802.1AE ([MACSEC]) because that is also supported in their chips.
   Only a flexible gateway device may need to support both of these
   mechanisms and potentially more.

   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:

   o 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.

   o Once the first secure channel protocol succeeds, the two peers
      know each other's certificates because they 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.

   o 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 be 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 connection from Bob succeeded, Alice
might prefer another secure protocol over IPsec (e.g., FOOBAR), and
try to set that up with Bob. If that preference of Alice 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. Attempts to
reconnect MUST be throttled. The RECOMMENDED default is exponential
backoff with a minimum delay of 10 seconds and a maximum delay of 640
seconds.

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 that is supporting native
   IPsec MUST use IPsec security setup via IKEv2, tunnel mode, local and
   peer link-local IPv6 addresses used for encapsulation. It MUST then
   support ESP with AES256 for encryption and SHA256 hash and MUST NOT
   permit weaker crypto options.

   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. An ACP node that is supporting ACP via GRE/IPsec MUST then
   support IPsec security setup via IKEv2, IPsec transport mode, local
   and peer link-local IPv6 addresses used for encapsulation, 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 MUST 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

As explained in the beginning of Section 6.5, there is no single secure channel mechanism mandated for all ACP nodes. Instead, this section defines two ACP profiles (baseline and constrained) for ACP nodes that do introduce such requirements.

A baseline ACP node MUST support IPsec natively and MAY support IPsec via GRE. A constrained ACP node that cannot support IPsec MUST support DTLS. An ACP node connecting an area of constrained ACP nodes with an area of baseline ACP nodes MUST therefore support IPsec and DTLS and supports therefore the baseline and constrained profile.

ACP nodes need to specify in documentation the set of secure ACP mechanisms they support and should declare which profile they support according to above requirements.

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. The function in GRASP that makes it fundamental
   as a service of the ACP is the ability to provide ACP wide service
   discovery (using objectives in GRASP).

   ACP provides IP unicast routing via the RPL routing protocol (see
   Section 6.11).

   The ACP does not use IP multicast routing nor does it provide generic
   IP multicast services (the handling of GRASP link-local multicast
   messages is explained in Section 6.8.2). Instead, the ACP provides
   service discovery via the objective discovery/announcement and
   negotiation mechanisms of the ACP GRASP instance (services are a form
   of objectives). These mechanisms use hop-by-hop reliable flooding of
   GRASP messages for both service discovery (GRASP M_DISCOVERY
   messages) and service announcement (GRASP M_FLOOD messages).

   See Appendix A.5 for discussion about this design choice of the ACP.

6.8.2. 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 for ensuring that this
   instance of GRASP is only sending messages across the ACP GRASP
   virtual interfaces. Whenever the ACP adds or deletes such an
   interface because of new ACP secure channels or loss thereof, the ACP
   needs to indicate this to the ACP instance of GRASP. The ACP exists
   also in the absence of any active ACP neighbors. It is created when
   the node has a domain certificate, and continues to exist even if all
   of its neighbors cease operation.

   In this case ASAs using GRASP running on the same node would still
   need to be able to discover each other's objectives. When the ACP
   does not exist, ASAs leveraging the ACP instance of GRASP via APIs
   MUST still be able to operate, and MUST be able to understand that
   there is no ACP and that therefore the ACP instance of GRASP cannot
   operate.

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

```
...............................ACP.............................
.                                                             .
.         /-GRASP-flooding-\          ACP GRASP instance      .
.        /                  \                                 A
.     GRASP      GRASP      GRASP                             C
.   link-local  unicast   link-local                         P
.   multicast   messages  multicast                          .
.   messages      |       messages                           .
.       |         |          |                               .
...............................................................
.       v         v          v   ACP security and transport  .
.       |         |          |      substrate for GRASP      .
.       |         |          |                               .
.       |     ACP GRASP      |      - ACP GRASP              A
.       |     Loopback       |        Loopback interface     C
.       |     interface      |      - ACP-cert auth          P
.       |        TLS         |                               .
.   ACP GRASP     |      ACP GRASP  - ACP GRASP virtual      .
.   subnet1       |      subnet2      virtual interfaces      .
.    TCP          |       TCP                                .
.       |         |          |                               .
...............................................................
.       |         |          |   ^^^ Users of ACP (GRASP/ASA) .
.       |         |          |   ACP interfaces/addressing    .
.       |         |          |                               .
.       |         |          |                               A
.       | ACP-Loopback Interf.|     <- ACP Loopback interface C
.       |       ACP-address   |      - address (global ULA)  P
.    subnet1     |         subnet2 <- ACP virtual interfaces .
.   link-local   |        link-local - link-local addresses  .
...............................................................
.       |         |          |   ACP routing and forwarding  .
.       |      RPL-routing    |                               .
.       |     /IP-Forwarding\ |                               A
.       |    /             \  |                               C
.   ACP IPv6 packets    ACP IPv6 packets                     P
.       |/              \|                                   .
.    IPsec/DTLS       IPsec/DTLS  - ACP-cert auth            .
...............................................................
        |                |   Data-Plane
        |                |
        |                |      - ACP secure channel
    link-local       link-local - encapsulation addresses
      subnet1          subnet2  - Data-Plane interfaces
        |                |
    ACP-Nbr1         ACP-Nbr2
```

Figure 7: ACP as security and transport substrate for GRASP

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 ACP GRASP virtual interface that is constructed from the TCP
connection(s) to the IPv6 link-local neighbor address(es) on the
underlying ACP virtual interface. If the ACP GRASP virtual interface
has two or more neighbors, the GRASP link-local multicast messages
are replicated to all neighbor TCP connections.

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. Such shorter periodic retransmission of datagrams would
result in more traffic and processing overhead in the ACP than the
hop-by-hop reliable retransmission mechanism by TCP and duplicate
elimination by GRASP.

TLS is mandated for GRASP non-link-local unicast because the ACP
secure channel mandatory authentication and encryption protects only
against attacks from the outside but not against attacks from the
inside: Compromised ACP members that have (not yet) been detected and
removed (e.g., via domain certificate revocation / expiry).

If GRASP peer connections 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 semantics of the GRASP
negotiation to an extent that allows it to proxy it without being
detected, but in an ACP environment this is quite likely public
knowledge or even standardized.

The GRASP TLS connections are run the same as any other ACP traffic
through the ACP secure channels. This leads to double
authentication/encryption, which has the following benefits:

o Secure channel methods such as IPsec may provide protection
   against additional attacks, for example reset-attacks.

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

o There is no different security model for ACP GRASP from other ACP
   traffic. Instead, there is just another layer of protection
   against certain attacks from the inside which is important due to
   the role of GRASP in the ACP.

6.9. 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 system, 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 Data-Plane routing or forwarding table(s). 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 prefix. Conversely, ULA global IDs
   from different domains are unlikely to clash, such that two ACP
   networks can be merged, as long as the policy allows that merge. See
   also Section 9.1 for a discussion on merging domains.

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

6.10.1. Fundamental Concepts of Autonomic Addressing

   o 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).

   o Separation: Autonomic address space is used separately from user
      address space and other address realms. This supports the
      robustness requirement.

   o 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].

   o Use-ULA: For Loopback interfaces of ACP nodes, we use Unique Local
      Addresses (ULA), as defined in [RFC4193] with L=1 (as defined in
      section 3.1 of [RFC4193]). Note that the random hash for ACP
      Loopback addresses uses the definition in Section 6.10.2 and not
      the one of [RFC4193] section 3.2.2.

   o 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.

o Addresses in the ACP are permanent, and do not support temporary
   addresses as defined in [RFC4941].

o 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:

o 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.

o 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.

o OAM protocols do not require IPv4: The ACP may carry OAM
   protocols. All relevant protocols (SNMP, TFTP, SSH, SCP, Radius,
   Diameter, ...) are available in IPv6. See also [RFC8368] for how
   ACP could be made to interoperate with IPv4 only OAM.

6.10.2. The ACP Addressing Base Scheme

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

```
  8        40                      2                 78
+--+-------------------------+------+----------------------------+
|fd| hash(routing-subdomain) | Type |      (sub-scheme)          |
+--+-------------------------+------+----------------------------+
```
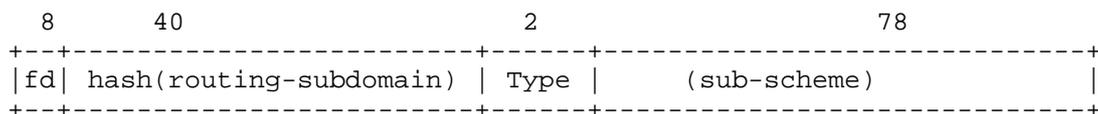
                  Figure 8: ACP Addressing Base Scheme

The first 48-bits follow the ULA scheme, as defined in [RFC4193], to
which a type field is added:

o "fd" identifies a locally defined ULA address.

o 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"
89b714f3db.

o To allow for extensibility, the fact that the ULA "global ID" is a
   hash of the routing subdomain SHOULD NOT be assumed by any ACP
   node during normal operations. The hash function is only executed
   during the creation of the certificate. If BRSKI is used then the
   BRSKI registrar will create the domain information field in
   response to the EST Certificate Signing Request (CSR) Attribute
   Request message by the pledge.

o Type: This field allows different address sub-schemes. This
   addresses the "upgradability" requirement. Assignment of types
   for this field will be maintained by IANA.

The sub-scheme may imply a range or set of addresses assigned to the
node, this is called the ACP address range/set and explained in each
sub-scheme.

Please refer to Section 6.10.7 and Appendix A.1 for further
explanations why the following Sub-Addressing schemes are used and
why multiple are necessary.

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 and 0 in the Z bit.

```
                 64                            64
+-----------------+---+---------++---------------------------+---+
|  (base scheme)  | Z | Zone-ID ||          Node-ID          |   |
|                 |   |         || Registrar-ID | Node-Number| V |
+-----------------+---+---------++--------------+------------+---+
        50          1     13          48             15       1
```

Figure 9: ACP Zone Addressing Sub-Scheme

The fields are defined as follows:

o 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.

o Z: MUST be 0.

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

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

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

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

o V (1-bit): Virtualization bit: 0: Indicates the ACP itself ("ACP
   node base system); 1: Indicates the optional "host" context on the
   ACP node (see below).

In the ACP Zone Addressing Sub-Scheme, the ACP address in the
certificate has Zone-ID and V fields as all zero bits. The ACP
address set includes addresses with any Zone-ID 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). An address with Zone-ID = 0 is an
identifier, with a Zone-ID !=0 it is a locator. See Section 6.10.3.1
for more details.

The Virtual bit in this sub-scheme allows the easy addition of the
ACP as a component to existing systems without causing problems in
the port number space between the services in the ACP and the
existing system. V:0 is the ACP router (autonomic node base system),
V:1 is the host with pre-existing transport endpoints on it that
could collide with the transport endpoints used by the ACP router.
The ACP host could for example have a p2p virtual interface with the
V:0 address as its router into the ACP. Depending on the software
design of ASAs, which is outside the scope of this specification,
they may use the V:0 or V:1 address.

The location of the V bit(s) at the end of the address allows the
announcement of a single prefix for each ACP node. For example, in a
network with 20,000 ACP nodes, this avoid 20,000 additional routes in
the routing table.

6.10.3.1. Usage of the Zone-ID Field

   The Zone-ID allows for the introduction of route prefixes in the
   addressing scheme.

   Zone-ID = 0 is the default addressing scheme in an ACP domain. Every
   ACP node with a Zone Addressing Sub-Scheme address MUST respond to
   its ACP address with Zone-ID = 0. Used on its own this leads to a
   non-hierarchical address scheme, which is suitable for networks up to
   a certain size. Zone-ID = 0 addresses act as identifiers for the
   nodes, and aggregation of these address in the ACP routing table is
   not possible.

   If aggregation is required, the 13-bit Zone-ID value allows for up to
   8191 zones. The allocation of Zone-ID's may either happen
   automatically through a to-be-defined algorithm; or it could be
   configured and maintained explicitly.

   If a node learns (see Appendix A.10.1) that it is part of a zone, it
   MUST also respond to its ACP address with that Zone-ID. In this case
   the ACP Loopback is configured with two ACP addresses: One for Zone-
   ID = 0 and one for the assigned Zone-ID. This method allows for a
   smooth transition between a flat addressing scheme and a hierarchical
   one.

   A node knowing it is in a zone MUST also use that Zone-ID != 0
   address in GRASP locator fields. This eliminates the use of the
   identifier address (Zone-ID = 0) in forwarding and the need for
   network wide reachability of those non-aggregable identifier
   addresses. Zone-ID != 0 addresses are assumed to be aggregable in
   routing/forwarding based on how they are allocated in the ACP
   topology.

   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 multiple /48 Global IDs within an ACP
   domain.

   Note: Zones and Zone-ID as defined here are not related to [RFC4007]
   zones or zone_id. ACP zone addresses are not scoped (reachable only
   from within an RFC4007 zone) but reachable across the whole ACP. An
   RFC4007 zone_id is a zone index that has only local significance on a
   node, whereas an ACP Zone-ID is an identifier for an ACP zone that is
   unique across that ACP.

6.10.4. ACP Manual Addressing Sub-Scheme

   The sub-scheme defined here is defined by the Type value 00b (zero)
   in the base scheme and 1 in the Z bit.


```
                  64                                  64
    +--------------------+---+---------++----------------------------+
    |   (base scheme)    | Z | Subnet-ID||    Interface Identifier   |
    +--------------------+---+---------++----------------------------+
            50             1    13
```
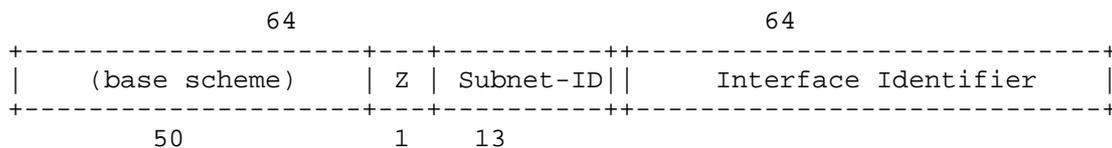
                Figure 10: ACP Manual Addressing Sub-Scheme

   The fields are defined as follows:

   o Subnet-ID: Configured subnet identifier.

   o Z: MUST be 1.

   o Interface Identifier.

   This sub-scheme is meant for "manual" allocation to subnets where the
   other addressing schemes cannot be used. The primary use case is for
   assignment to ACP connect subnets (see Section 8.1.1).

   "Manual" means that allocations of the Subnet-ID need to be done
   today with pre-existing, non-autonomic mechanisms. Every subnet that
   uses this addressing sub-scheme needs to use a unique Subnet-ID
   (unless some anycast setup is done).

   The Z bit field was added to distinguish Zone addressing and manual
   addressing sub-schemes without requiring one more bit in the base
   scheme and therefore allowing for the Vlong scheme (described below)
   to have one more bit available.

   Manual addressing sub-scheme addresses SHOULD NOT be used in ACP
   domain certificates. Any node capable to build ACP secure channels
   and permitted by Registrar policy to participate in building ACP
   secure channels SHOULD receive an ACP address (prefix) from one of
   the other ACP addressing sub-schemes. Nodes not capable (or
   permitted) to participate in ACP secure channels can connect to the
   ACP via ACP connect interfaces of ACP edge nodes (see Section 8.1),
   without setting up an ACP secure channel. Their ACP domain
   certificate MUST include an empty acp-address to indicate that their
   ACP domain certificate is only usable for non- ACP secure channel

authentication, such as end-to-end transport connections across the ACP or Data-Plane.

Address management of ACP connect subnets is done using traditional assignment methods and existing IPv6 protocols. See Section 8.1.3 for details.

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.

```
           50                                78
 +----------------------++---------------------------+----------+
 |     (base scheme)    ||           Node-ID         |          |
 |                      || Registrar-ID |  Node-Number|     V  |
 +----------------------++--------------+-------------+----------+
           50                46            24/16        8/16
```
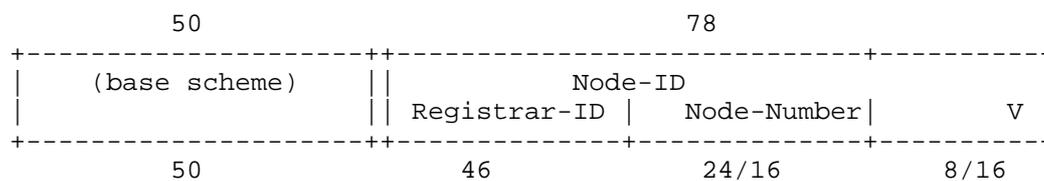
                Figure 11: ACP Vlong Addressing Sub-Scheme

This addressing scheme foregoes the Zone-ID field to allow for larger, flatter routed networks (e.g., as in IoT) with 8421376 Node-Numbers ($2^{23}+2^{15}$). It also allows for up to 2^{16} (i.e. 65536) different virtualized addresses within a node, which could be used to address individual software components in an ACP node.

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

o V: Virtualization bit: Values 0 and 1 are assigned in the same way as in the Zone-ID sub-scheme.

o Registrar-ID: To maximize Node-Number and V, the Registrar-ID is reduced to 46-bits. This still permits the use of the MAC address of an ACP 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).

o If the first bit of the "Node-Number" is "1", then the Node-Number is 16-bit long and the V field is 16-bit long. Otherwise the Node-Number is 24-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) of 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-Scheme is intended to enable optimized routing in
large networks by reserving bits for Zone-ID's. The Vlong addressing
sub-scheme enables the allocation of 8/16-bit of addresses inside
individual ACP nodes. Both address spaces allow distributed,
uncoordinated allocation of node addresses by reserving bits for the
registrar-ID field in the address.

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 MUST provide further
extensions (e.g., by reserving bits from it for further extensions).

6.10.7. ACP Registrars

The ACP address prefix is assigned to the ACP node during enrollment/
provisioning of the ACP domain certificate to the ACP node. It is
intended to persist unchanged through the lifetime of the ACP node.

Because of the ACP addressing sub-schemes explained above, ACP nodes
for a single ACP domain can be enrolled by multiple distributed and
uncoordinated entities called ACP registrars. These ACP registrars
are responsible to enroll ACP domain certificates and associated
trust anchor(s) to candidate ACP nodes and are also responsible that
an ACP domain information field is included in the ACP domain
certificate.

6.10.7.1. Use of BRSKI or other Mechanism/Protocols

Any protocols or mechanisms may be used as ACP registrars, as long as
the resulting ACP certificate and trust anchors allow to perform the
ACP domain membership described in Section 6.1.2 with other ACP

domain members, and meet the ACP addressing requirements for its ACP
domain information field as described further below in this section.

An ACP registrar could be a person deciding whether to enroll a
candidate ACP node and then orchestrating the enrollment of the ACP
certificate and associated trust anchor, using command line or web
based commands on the candidate ACP node and trust anchor to generate
and sign the ACP domain certificate and configure certificate and
trust anchors onto the node.

The only currently defined protocol for ACP registrars is BRSKI
([I-D.ietf-anima-bootstrapping-keyinfra]). When BRSKI is used, the
ACP nodes are called ANI nodes, and the ACP registrars are called
BRSKI or ANI registrars. The BRSKI specification does not define the
handling of the ACP domain information field because the rules do not
depend on BRSKI but apply equally to any protocols/mechanisms an ACP
registrar may use.

6.10.7.2. Unique Address/Prefix allocation

ACP registrars MUST NOT allocate ACP address prefixes to ACP nodes
via the ACP domain information field that would collide with the ACP
address prefixes of other ACP nodes in the same ACP domain. This
includes both prefixes allocated by the same ACP registrar to
different ACP nodes as well as prefixes allocated by other ACP
registrars for the same ACP domain.

For this purpose, an ACP registrar MUST have one or more unique
46-bit identifiers called Registrar-IDs used to allocate ACP address
prefixes. The lower 46-bits of a EUI-48 MAC addresses are globally
unique 46 bit identifiers, so ACP registrars with known unique EUI-48
MAC addresses can use these as Registrar-IDs. Registrar-IDs do not
need to be globally unique but only unique across the set of ACP
registrars for an ACP domain, so other means to assign unique
Registrar-IDs to ACP registrars can be used, such as configuration on
the ACP registrars.

When the candidate ACP device (called Pledge in BRSKI) is to be
enrolled into an ACP domain, the ACP registrar needs to allocate a
unique ACP address to the node and ensure that the ACP certificate
gets a domain information field (Section 6.1.1) with the appropriate
information - ACP domain-name, ACP-address, and so on. If the ACP
registrar uses BRSKI, it signals the ACP information field to the
Pledge via the EST /csraddrs command (see
[I-D.ietf-anima-bootstrapping-keyinfra], section 5.8.2 - "EST CSR
Attributes").

   [RFC Editor: please update reference to section 5.8.2 accordingly
   with latest BRSKI draft at time of publishing, or RFC]

6.10.7.3. Addressing Sub-Scheme Policies

   The ACP registrar selects for the candidate ACP node a unique address
   prefix from an appropriate ACP addressing sub-scheme, either a zone
   addressing sub-scheme prefix (see Section 6.10.3), or a Vlong
   addressing sub-scheme prefix (see Section 6.10.5). The assigned ACP
   address prefix encoded in the domain information field of the ACP
   domain certificate indicates to the ACP node its ACP address
   information. The sub-addressing scheme indicates the prefix length:
   /127 for zone address sub-scheme, /120 or /112 for Vlong address sub-
   scheme. The first address of the prefix is the ACP address, all
   other addresses in the prefix are for other uses by the ACP node as
   described in the zone and Vlong addressing sub scheme sections. The
   ACP address prefix itself is then signaled by the ACP node into the
   ACP routing protocol (see Section 6.11) to establish IPv6
   reachability across the ACP.

   The choice of addressing sub-scheme and prefix-length in the Vlong
   address sub-scheme is subject to ACP registrar policy. It could be
   an ACP domain wide policy, or a per ACP node or per ACP node type
   policy. For example, in BRSKI, the ACP registrar is aware of the
   IDevID of the candidate ACP node, which contains a serialNnumber that
   is typically indicating the nodes vendor and device type and can be
   used to drive a policy selecting an appropriate addressing sub-scheme
   for the (class of) node(s).

   ACP registrars SHOULD default to allocate ACP zone sub-address scheme
   addresses with Subnet-ID 0. Allocation and use of zone sub-addresses
   with Subnet-ID != 0 is outside the scope of this specification
   because it would need to go along with rules for extending ACP
   routing to multiple zones, which is outside the scope of this
   specification.

   ACP registrars that can use the IDevID of a candidate ACP device
   SHOULD be able to choose the zone vs. Vlong sub-address scheme for
   ACP nodes based on the serialNumber of the IDevID, for example by the
   PID (Product Identifier) part which identifies the product type, or
   the complete serialNumber.

   In a simple allocation scheme, an ACP registrar remembers
   persistently across reboots for its currently used Registrar-ID and
   for each addressing scheme (zone with Subnet-ID 0, Vlong with /112,
   Vlong with /120), the next Node-Number available for allocation and
   increases it after successful enrollment to an ACP node. In this

simple allocation scheme, the ACP registrar would not recycle ACP
address prefixes from no longer used ACP nodes.

6.10.7.4. Address/Prefix Persistence

When an ACP domain certificate is renewed or rekeyed via EST or other
mechanisms, the ACP address/prefix in the ACP domain information
field MUST be maintained unless security issues or violations of the
unique address assignment requirements exist or are suspected by the
ACP registrar. Even when the renewing/rekeying ACP registrar is not
the same as the one that enrolled the prior ACP certificate. See
Section 10.2.4 for an example. ACP address information SHOULD also
be maintained even after an ACP certificate did expire or failed.
See Section 6.1.3.5 and Section 6.1.3.6.

6.10.7.5. Further Details

Section 10.2 discusses further informative details of ACP registrars:
What interactions registrars need, what parameters they require,
certificate renewal and limitations, use of sub-CAs on registrars and
centralized policy control.

6.11. 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 Data-Plane routing tables
and also secures the ACP from interference from the configuration
mismatch or incorrect routing updates.

The establishment of the routing plane and its parameters are
automatic and strictly within the confines of the autonomic control
plane. Therefore, no explicit configuration is required.

All routing updates are automatically secured in transit as the
channels of the 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
Appendix A.4 for more details on the choice of RPL.

RPL adjacencies are set up across all ACP channels in the same domain
including all its routing subdomains. See Appendix A.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 with reasonably 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 forwarding
   plane, especially to support devices with silicon forwarding planes
   that cannot support insertion/removal of these headers in 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 that is containing
   one Destination Oriented Directed Acyclic Graph (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 will reconfigure itself when it detects data plane forwarding
   failures. See Appendix A.10.4 for more details.

   The lack of RPL Packet Information (RPI, the IPv6 header for RPL
   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 time-to-live (TTL) of the packet reaches zero. This
   the same behavior as that of other IGPs that do not have the Data-
   Plane options as RPL.

   Since links in the ACP are assumed to be mostly reliable (or have
   link layer protection against loss) and because there is no stretch

according to Section 6.11.1.7, loops should be exceedingly rare though.

There are a variety of mechanisms possible in RPL to further avoid temporary loops: DODAG Information Objects (DIOs) SHOULD be sent 2...3 times to inform children when losing the last parent. The technique in [RFC6550] section 8.2.2.6. (Detaching) SHOULD be favored over that in section 8.2.2.5., (Poisoning) because it allows local connectivity. Nodes SHOULD select more than one parent, at least 3 if possible, and send Destination Advertisement Objects (DAO)s to all of them in parallel.

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

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): MUST support mode 2 - "Storing Mode of Operations with no multicast support". Implementations MAY support mode 3 ("... with multicast support" as that is a superset of mode 2). Note: Root indicates mode in DIO flow.

6.11.1.4. DAO Policy

Proactive, aggressive DAO state maintenance:

o Use K-flag in unsolicited DAO indicating change from previous information (to require DAO-ACK).

o 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 ([RFC6550], 3.2.6 - to become root):
   Indicated in DODAGPreference field of DIO message.

   o Explicit configured "root": 0b100

   o ACP registrar (Default): 0b011

   o ACP-connect (non-registrar): 0b010

   o 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. Implementations
   of ACP to solely support traditional/SDN style use cases can benefit
   from ACP at lower performance, especially if the ACP is used only for
   critical operations, e.g., when the Data-Plane is not available. The
   design of the ACP as specified in this document is intended to

support a wide range of performance options: It is intended to allow
software-only implementations at potentially low performance, but can
also support high performance options. See [RFC8368] for more
details.

6.12.2. Addressing of Secure Channels

In order to be independent of the Data-Plane (routing and addressing)
the GRASP discovered (autonomic) ACP secure channels use IPv6 link
local addresses between adjacent neighbors. Note: Section 8.2
specifies extensions in which secure channels are configured tunnels
operating over the Data-Plane, so those secure channels cannot be
independent of the Data-Plane.

To avoid that Data-Plane configuration can impact the operations of
the IPv6 (link-local) interface/address used for ACP channels,
appropriate implementation considerations are required. If the IPv6
interface/link-local address is shared with the Data-Plane it needs
to be impossible to unconfigure/disable it through configuration.
Instead of sharing the IPv6 interface/link-local address, a separate
(virtual) interface with a separate IPv6 link-local address can be
used. For example, the ACP interface could be run over a separate
MAC address of an underlying L2 (Ethernet) interface. For more
details and options, see Appendix A.10.2.

Note that other (non-ideal) implementation choices may introduce
additional undesired dependencies against the Data-Plane. For
example shared code and configuration of the secure channel protocols
(IPsec / DTLS).

6.12.3. 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 an issue of
tunnels, not an issue of running the ACP across a tunnel. Transport
stacks running across ACP can perform normal PMTUD (Path MTU
Discovery). Because the ACP is meant to 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 were 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).

   Any type of ACP secure channels to another ACP node can be mapped to
   ACP virtual interfaces in following ways. This is independent of the
   chosen secure channel protocol (IPsec, DTLS or other future protocol
   - standards or non-standards):

   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.

"Optimistic Duplicate Address Detection (DAD)" according to [RFC4429]
is RECOMMENDED because the likelihood for duplicates between ACP
nodes is highly improbable as long as the address can be formed from
a globally unique local assigned identifier (e.g., EUI-48/EUI-64, see
below).

ACP nodes MAY reduce the amount of link-local IPv6 multicast packets
from ND by learning the IPv6 link-local neighbor address to ACP
secure channel mapping from other messages such as the source address
of IPv6 link-local multicast RPL messages - and therefore forego the
need to send Neighbor Solicitation messages.

The ACP virtual interface IPv6 link local address can be derived from
any appropriate local mechanism such as node local EUI-48 or EUI-64
("EUI" stands for "Extended Unique Identifier"). It MUST NOT depend
on something that is attackable from the Data-Plane such as the IPv6
link-local address of the underlying physical interface, which can be

attacked by SLAAC, or parameters of the secure channel encapsulation header that may not be protected by the secure channel mechanism.

The link-layer address of an ACP virtual interface is the address used for the underlying interface across which the secure tunnels are built, typically Ethernet addresses. Because unicast IPv6 packets sent to an ACP virtual interface are not sent to a link-layer destination address but rather an ACP secure channel, the link-layer address fields SHOULD be ignored on reception and instead the ACP secure channel from which the message was received should be remembered.

Multi-access ACP virtual interfaces are preferable implementations when the underlying interface is a (broadcast) multi-access subnet because they do reflect the presence of the underlying multi-access subnet into the virtual interfaces of the ACP. This makes it for example simpler to build services with topology awareness inside the ACP VRF in the same way as they could have been built running natively on the multi-access interfaces.

Consider also the impact of point-to-point vs. multi-access virtual interface on the efficiency of flooding via link local multicasted messages:

Assume a LAN with three ACP neighbors, Alice, Bob and Carol. Alice's ACP GRASP wants to send a link-local GRASP multicast message to Bob and Carol. If Alice's ACP emulates the LAN as 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 Charlie's ACP would emulate a multi-access virtual interface, then this would not happen, because GRASPs flooding procedure does not replicate back packets to the interface that they were received from.

Note that link-local GRASP multicast messages are not sent directly as IPv6 link-local multicast UDP messages into ACP virtual interfaces, but instead into ACP GRASP virtual interfaces, that are layered on top of ACP virtual interfaces to add TCP reliability to link-local multicast GRASP messages. Nevertheless, these ACP GRASP virtual interfaces perform the same replication of message and, therefore, result in the same impact on flooding. See Section 6.8.2 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. See also Appendix A.10.4.

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

```
    ANrtr1 ------ ANswitch1 --- ANswitch2 ------- ANrtr2
         .../    \                       \  ...
    ANrtrM ------      \                     ------- ANrtrN
                    ANswitchM ...
```
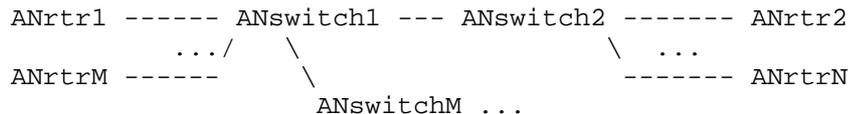
Figure 12: Topology with L2 ACP switches

Consider a large L2 LAN with ANrtr1...ANrtrN connected via some topology of L2 switches. Examples include large enterprise campus networks with an L2 core, IoT networks or broadband aggregation networks which often have even a multi-level L2 switched topology.

If the discovery protocol used for the ACP is operating at the subnet level, every ACP router will see all other ACP routers on the LAN as neighbors and a full mesh of ACP channels will be built. If some or all of the AN switches are autonomic with the same discovery protocol, then the full mesh would include those switches as well.

A full mesh of ACP connections can create fundamental scale challenges. The number of security associations of the secure channel protocols will likely not scale arbitrarily, especially when they leverage platform accelerated encryption/decryption. Likewise, any other ACP operations (such as routing) needs to scale to the number of direct ACP neighbors. An ACP router with just 4 physical interfaces might be deployed into a LAN with hundreds of neighbors connected via switches. Introducing such a new unpredictable scaling

factor requirement makes it harder to support the ACP on arbitrary platforms and in arbitrary deployments.

Predictable scaling requirements for ACP neighbors can most easily be achieved if in topologies such as these, ACP capable L2 switches can ensure that discovery messages terminate on them so that neighboring ACP routers and switches will only find the physically connected ACP L2 switches as their candidate ACP neighbors. With such a discovery mechanism in place, the ACP and its security associations will only need to scale to the number of physical interfaces instead of a potentially much larger number of "LAN-connected" neighbors. And the ACP topology will follow directly the physical topology, something which can then also be leveraged in management operations or by ASAs.

In the example above, consider ANswitch1 and ANswitchM are ACP capable, and ANswitch2 is not ACP capable. The desired ACP topology is that ANrtr1 and ANrtrM only have an ACP connection to ANswitch1, and that ANswitch1, ANrtr2, ANrtrN have a full mesh of ACP connection amongst each other. ANswitch1 also has an ACP connection with ANswitchM and ANswitchM has ACP connections to anything else behind it.

7.2. How (per L2 port DULL GRASP)

To support ACP on L2 switches or L2 switched ports of an L3 device, it is necessary to make those L2 ports look like L3 interfaces for the ACP implementation. This primarily involves the creation of a separate DULL GRASP instance/domain on every such L2 port. Because GRASP has a dedicated link-local IPv6 multicast address (ALL_GRASP_NEIGHBORS), it is sufficient that all packets for this address are being extracted at the port level and passed to that DULL GRASP instance. Likewise the IPv6 link-local multicast packets sent by that DULL GRASP instance need to be sent only towards the L2 port for this DULL GRASP instance.

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.

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, such as any other external node.

   If the NMS host is not autonomic, i.e., it does not support autonomic
   negotiation of the ACP, then it can be brought into the ACP by
   explicit configuration. To support connections to adjacent non-ACP
   nodes, an ACP node must support "ACP connect" (sometimes also called
   "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).

```
                           Data-Plane "native" (no ACP)
                                            .
+--------+       +---------------+          .        +-------------+
| ACP    |       |ACP Edge Node  |          .        |             |
| Node   |       |               |          v        |             |
|        |-------|...[ACP VRF]....+-----------------|             |+
|        |  ^    |.              |                   | NOC Device  ||
|        |  .    | .[Data-Plane]..+-----------------| "NMS hosts" ||
|        |  .    | [         ] | .         ^         |             ||
+--------+  .    +---------------+ .        .        +-------------+|
            .                      .        .          +------------+
            .   Data-Plane "native" .    ACP "native" (unencrypted)
            + ACP auto-negotiated   .    "ACP connect subnet"
              and encrypted         .
                              ACP connect interface
                              e.g., "VRF ACP native" (config)
```
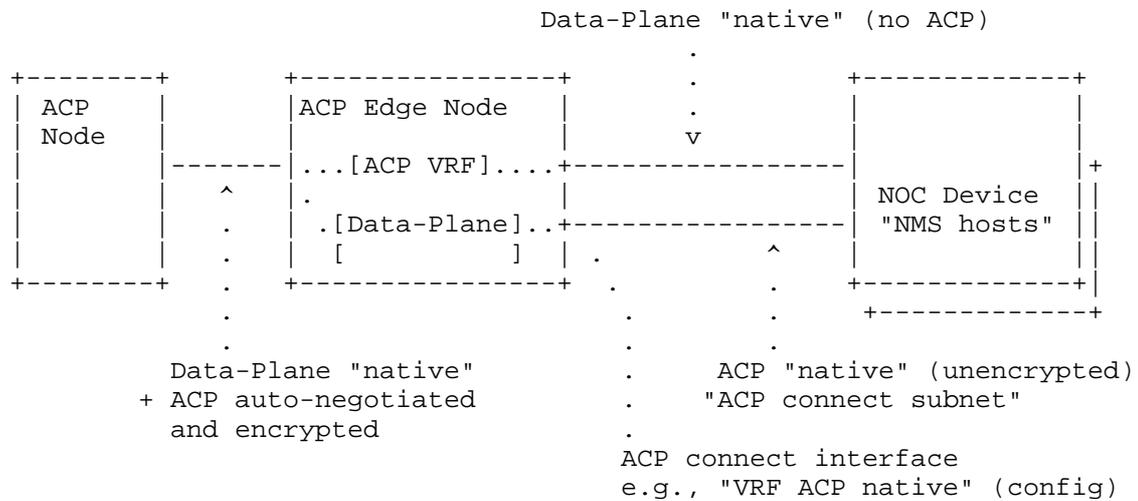
                        Figure 13: ACP connect

   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
   "Using Autonomic Control Plane for Stable Connectivity of Network
   OAM" [RFC8368] explains in more detail how the ACP can be integrated
   in a mixed NOC environment.

   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 (those in physically secure locations such as a NOC). For example, the registrar could disable the ability to enable ACP connect on devices during enrollment and that property could only be changed through re-enrollment. See also Appendix A.10.5.

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 for ensuring that only trusted
software components will get access to that virtual subnet and that
only even more trusted software components will get access to both
the ACP virtual subnet and the Data-Plane (because those ACP users
could leak traffic between ACP and Data-Plane). This trust could be
established for example through cryptographic means such as signed
software packages.

8.1.3. Auto Configuration

   ACP edge nodes, NMS hosts and software components that as described
   in the previous section are meant to be composed via virtual
   interfaces SHOULD support on the ACP connect subnet StateLess Address
   Autoconfiguration (SLAAC - [RFC4862]) and route auto configuration
   according to [RFC4191].

   The ACP edge node acts as the router 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 (ACP Zone Addressing Sub-
   Scheme), /112 or /120 (Vlong Addressing Sub-Scheme) routes of actual
   ACP nodes. If ACP interfaces are configured with non ULA prefixes,
   then those prefixes cannot be aggregated without further configured
   policy on the ACP edge node. This explains the above recommendation
   to use ACP ULA prefix covered prefixes for ACP connect interfaces:
   They allow for a shorter list of prefixes to be signaled via RFC4191
   to NMS hosts and software components.

   The ACP edge nodes that have a Vlong ACP address MAY allocate a
   subset of their /112 or /120 address prefix to ACP connect
   interface(s) to eliminate the need to non-autonomically configure/
   provision the address prefixes for such ACP connect interfaces.

8.1.4. Combined ACP/Data-Plane Interface (VRF Select)

```
                       Combined ACP and Data-Plane interface
                                            .
   +--------+        +--------------------+  .  +--------------+
   | ACP    |        |ACP Edge No         |  .  | NMS Host(s)  |
   | Node   |        |                    |  .  | / Software   |
   |        |        |  [ACP  ].          |  .  |             |+
   |        |        | .[VRF  ] .[VRF   ] |  v  | "ACP address"||
   |        +-------+.            .[Select].+--------+ "Date Plane  ||
   |        |   ^   | .[Data ].          |     |   Address(es)"||
   |        |   .   | [Plane]            |     |             ||
   |        |   .   | [      ]           |     +--------------+|
   +--------+   .   +--------------------+     +--------------+
               .
       Data-Plane "native" and + ACP auto-negotiated/encrypted
```
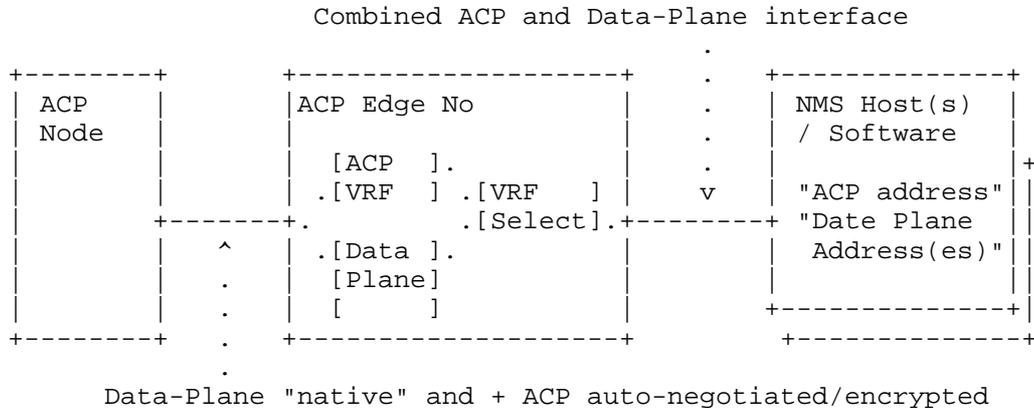
Figure 14: VRF select

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. 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. Hosts implementing [RFC8028] should (instead of may)
implement [RFC6724] Rule 5.5, so it is preferred for hosts to support
[RFC8028].

ACP edge nodes MAY support the Combined ACP and Data-Plane interface.

8.1.5. Use of GRASP

GRASP can and should be possible to use across ACP connect
interfaces, especially in the architectural correct solution when it
is used as a mechanism to connect Software (e.g., ASA or legacy NMS
applications) to the ACP. Given how the ACP is the security and
transport substrate for GRASP, the 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 the
same as 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 explicitly. The
parameters of such a "connection" are described in the following
ABNF.

```
connection = [ method , local-addr, remote-addr, ?pmtu ]
method   = [ "IKEv2" , ?port ]
method //= [ "DTLS",    port ]
local-addr  = [ address , ?vrf  ]
remote-addr = [ address ]
address = ("any" | ipv4-address | ipv6-address )
vrf = tstr ; Name of a VRF on this node with local-address
```

Figure 15: Parameters for remote ACP neighbors

Explicit configuration of a remote-peer according to this ABNF
provides all the information to build a secure channel without
requiring a tunnel to that peer and running DULL GRASP inside of it.

The configuration includes the parameters otherwise signaled via DULL
GRASP: local address, remote (peer) locator and method. The
differences over DULL GRASP local neighbor discovery and secure
channel creation are as follows:

o The local and remote address can be IPv4 or IPv6 and are typically
   global scope addresses.

o The VRF across which the connection is built (and in which local-
   addr exists) can to be specified. If vrf is not specified, it is
   the default VRF on the node. In DULL GRASP the VRF is implied by
   the interface across which DULL GRASP operates.

o If local address is "any", the local address used when initiating
   a secure channel connection is decided by source address selection
   ([RFC6724] for IPv6). As a responder, the connection listens on
   all addresses of the node in the selected VRF.

o Configuration of port is only required for methods where no
   defaults exist (e.g., "DTLS").

o If remote address is "any", the connection is only a responder.
   It is a "hub" that can be used by multiple remote peers to connect
   simultaneously - without having to know or configure their
   addresses. Example: Hub site for remote "spoke" sites reachable
   over the Internet.

o Pmtu should be configurable to overcome issues/limitations of Path
   MTU Discovery (PMTUD).

o IKEv2/IPsec to remote peers should support the optional NAT
   Traversal (NAT-T) procedures.

8.2.2. Tunneled Remote ACP Neighbor

   An IPinIP, GRE or other form of pre-existing tunnel is configured
   between two remote ACP peers and the virtual interfaces representing
   the tunnel are configured for "ACP enable". This will enable IPv6
   link local addresses and DULL on this tunnel. In result, the tunnel
   is used for normal "L2 adjacent" candidate ACP neighbor discovery
   with DULL and secure channel setup procedures described in this
   document.

   Tunneled Remote ACP Neighbor requires two encapsulations: the
   configured tunnel and the secure channel inside of that tunnel. This
   makes it in general less desirable than Configured Remote ACP
   Neighbor. Benefits of tunnels are that it may be easier to implement
   because there is no change to the ACP functionality - just running it

over a virtual (tunnel) interface instead of only native interfaces.
The tunnel itself may also provide PMTUD while the secure channel
method may not. Or the tunnel mechanism is permitted/possible
through some firewall while the secure channel method may not.

8.2.3. Summary

   Configured/Tunneled Remote ACP neighbors are less "indestructible"
   than L2 adjacent ACP neighbors based on link local addressing, since
   they depend on more correct Data-Plane operations, such as routing
   and global addressing.

   Nevertheless, these options may be crucial to incrementally deploy
   the ACP, especially if it is meant to connect islands across the
   Internet. Implementations SHOULD support at least Tunneled Remote
   ACP Neighbors via GRE tunnels - which is likely the most common
   router-to-router tunneling protocol in use today.

9. Benefits (Informative)

9.1. Self-Healing Properties

   The ACP is self-healing:

   o New neighbors will automatically join the ACP after successful
      validation and will become reachable using their unique ULA
      address across the ACP.

   o 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.

   o 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, an ACP 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 enrollment of new nodes during the
partition.

Highly resilient ACP designs can be built by using ACP registrars
with embedded sub-CA, as outlined in Section 10.2.4. As long as a
partition is left with one or more of such ACP registrars, it can
continue to enroll new candidate ACP nodes as long as the ACP
registrars sub-CA certificate does not expire. Because the ACP
addressing relies on unique Registrar-IDs, a later re-merge of
partitions will also not cause problems with ACP addresses assigned
during partitioning.

After a network partition, a re-merge will just establish the
previous status, certificates can be renewed, the CRL is available,
and new nodes can be enrolled everywhere. Since all nodes use the
same 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 such as DTLS or IPsec which provide

additional authentication during channel establishment, data
integrity and data confidentiality protection of data inside the ACP
and in addition, provide replay protection.

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 fail 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 ACP
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 be protected against attackers so that there is no
easy way to compromise them, or use them as a proxy for attacking
other devices across the ACP. For example, management plane
functions (transport ports) should only be reachable from the ACP but
not the Data-Plane. Especially for those management plane functions
that have no good protection by themselves because they do not have
secure end-to-end transport and to whom ACP does not only provides
automatic reliable connectivity but also protection against attacks.
Protection across all potential attack vectors is typically easier to
do in devices whose software is designed from the ground up with

security in mind than with legacy software based systems where the
ACP is added on as another feature.

As explained above, traffic across the ACP SHOULD still be end-to-end
encrypted whenever possible. This includes traffic such as GRASP,
EST and BRSKI inside the ACP. This minimizes man in the middle
attacks by compromised ACP group members. Such attackers cannot
eavesdrop or modify communications, they can just filter them (which
is unavoidable by any means).

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. ACP Operations (Informative)

The following sections document important operational aspects of the
ACP. They are not normative because they do not impact the
interoperability between components of the ACP, but they include
recommendations/requirements for the internal operational model
beneficial or necessary to achieve the desired use-case benefits of
the ACP (see Section 3).

o Section 10.1 describes recommended operator diagnostics
   capabilities of ACP nodes. The have been derived from diagnostic
   of a commercially available ACP implementation.

o Section 10.2 describes high level how an ACP registrar needs to
   work, what its configuration parameters are and specific issues
   impacting the choices of deployment design due to renewal and
   revocation issues. It describes a model where ACP Registrars have
   their own sub-CA to provide the most distributed deployment option
   for ACP Registrars, and it describes considerations for
   centralized policy control of ACP Registrar operations.

o Section 10.3 describes suggested ACP node behavior and operational
   interfaces (configuration options) to manage the ACP in so-called
   greenfield devices (previously unconfigured) and brownfield
   devices (preconfigured).

The recommendations and suggestions of this chapter were derived from
operational experience gained with a commercially available pre-
standard ACP implementation.

10.1. ACP (and BRSKI) Diagnostics

Even though ACP and ANI in general are taking out many manual
configuration mistakes through their automation, it is important to
provide good diagnostics for them.

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:

o 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.

o 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.

o 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 such as the
above is to represent the relevant information sequentially in
dependency order, so that the first non-expected/non-operational item
is the most likely root cause. Or just log/highlight that item. For
example:

Q: Is ACP operational to accept neighbor connections:

o Check if any potentially necessary configuration to make ACP/ANI
   operational are correct (see Section 10.3 for a discussion of such
   commands).

o 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).

o Does the node have keying material - domain certificate, trust
   anchors.

o If no keying material and ANI is supported/enabled, check the
   state of BRSKI (not detailed in this example).

o 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?

   o Was the ACP VRF successfully created?

   o Is ACP enabled on one or more interfaces that are up and running?

   If all this looks good, the ACP should be running locally "fine" -
   but we did not check any ACP neighbor relationships.

   Question: why does the node not create a working ACP connection to a
   neighbor on an interface?

   o Is the interface physically up? Does it have an IPv6 link-local
     address?

   o Is it enabled for ACP?

   o Do we successfully send DULL GRASP messages to the interface (link
     layer errors)?

   o 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.

   o Do we see the ACP objective in any DULL GRASP message from that
     interface? Diagnose the supported secure channel methods.

   o Do we know the MAC address of the neighbor with the ACP objective?
     If not, diagnose SLAAC/ND state.

   o When did we last attempt to build an ACP secure channel to the
     neighbor?

   o 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 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 BRSKI registrars to locations/
    nodes where information about those pledges is desired.

2. The Link Layer Discovery Protocol (LLDP, [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".

10.2. ACP Registrars

As described in Section 6.10.7, the ACP addressing mechanism is
designed to enable lightweight, distributed and uncoordinated ACP
registrars that are providing ACP address prefixes to candidate ACP

nodes by enrolling them with an ACP domain certificate into an ACP
domain via any appropriate mechanism/protocol, automated or not.

This section discusses informatively more details and options for ACP
registrars.

10.2.1. Registrar interactions

This section summarizes and discusses the interactions with other
entities required by an ACP registrar.

In a simple instance of an ACP network, no central NOC component
beside a trust anchor (root CA) is required. One or more
uncoordinated acting ACP registrar can be set up, performing the
following interactions:

To orchestrate enrolling a candidate ACP node autonomically, the ACP
registrar can rely on the ACP and use Proxies to reach the candidate
ACP node, therefore allowing minimum pre-existing (auto-)configured
network services on the candidate ACP node. BRSKI defines the BRSKI
proxy, a design that can be adopted for various protocols that
Pledges/candidate ACP nodes could want to use, for example BRSKI over
CoAP (Constrained Application Protocol), or proxying of Netconf.

To reach a trust anchor unaware of the ACP, the ACP registrar would
use the Data-Plane. ACP and Data-Plane in an ACP registrar could
(and by default should be) completely isolated from each other at the
network level. Only applications such as the ACP registrar would
need the ability for their transport stacks to access both.

In non-autonomic enrollment options, the Data-Plane between a ACP
registrar and the candidate ACP node needs to be configured first.
This includes the ACP registrar and the candidate ACP node. Then any
appropriate set of protocols can be used between ACP registrar and
candidate ACP node to discover the other side, and then connect and
enroll (configure) the candidate ACP node with an ACP domain
certificate. Netconf ZeroTouch ([I-D.ietf-netconf-zerotouch]) is an
example protocol that could be used for this. BRSKI using optional
discovery mechanisms is equally a possibility for candidate ACP nodes
attempting to be enrolled across non-ACP networks, such as the
Internet.

When candidate ACP nodes have secure bootstrap, such as BRSKI
Pledges, they will not trust to be configured/enrolled across the
network, unless being presented with a voucher (see [RFC8366])
authorizing the network to take possession of the node. An ACP
registrar will then need a method to retrieve such a voucher, either
offline, or online from a MASA (Manufacturer Authorized Signing

Authority). BRSKI and Netconf ZeroTouch are two protocols that
include capabilities to present the voucher to the candidate ACP
node.

An ACP registrar could operate EST for ACP certificate renewal and/or
act as a CRL Distribution point. A node performing these services
does not need to support performing (initial) enrollment, but it does
require the same above described connectivity as an ACP registrar:
via the ACP to ACP nodes and via the Data-Plane to the trust anchor
and other sources of CRL information.

10.2.2. Registrar Parameter

The interactions of an ACP registrar outlined Section 6.10.7 and
Section 10.2.1 above depend on the following parameters:

A URL to the trust anchor (root CA) and credentials so that the
ACP registrar can let the trust anchor sign candidate ACP member
certificates.

The ACP domain-name.

The Registrar-ID to use. This could default to a MAC address of
the ACP registrar.

For recovery, the next-useable Node-IDs for zone (Zone-ID=0) sub-
addressing scheme, for Vlong /112 and for Vlong /1120 sub-
addressing scheme. These IDs would only need to be provisioned
after recovering from a crash. Some other mechanism would be
required to remember these IDs in a backup location or to recover
them from the set of currently known ACP nodes.

Policies if candidate ACP nodes should receive a domain
certificate or not, for example based on the devices LDevID as in
BRSKI. The ACP registrar may have a whitelist or blacklist of
devices serialNumbers from their LDevID.

Policies what type of address prefix to assign to a candidate ACP
devices, based on likely the same information.

For BRSKI or other mechanisms using vouchers: Parameters to
determine how to retrieve vouchers for specific type of secure
bootstrap candidate ACP nodes (such as MASA URLs), unless this
information is automatically learned such as from the LDevID of
candidate ACP nodes (as defined in BRSKI).

10.2.3. Certificate renewal and limitations

   When an ACP node renews/rekeys its certificate, it may end up doing
   so via a different registrar (e.g., EST server) than the one it
   originally received its ACP domain certificate from, for example
   because that original ACP registrar is gone. The ACP registrar
   through which the renewal/rekeying is performed would by default
   trust the ACP domain information from the ACP nodes current ACP
   domain certificate and maintain this information so that the ACP node
   maintains its ACP address prefix. In EST renewal/rekeying, the ACP
   nodes current ACP domain certificate is signaled during the TLS
   handshake.

   This simple scenario has two limitations:

   1. The ACP registrars cannot directly assign certificates to nodes
       and therefore needs an "online" connection to the trust anchor
       (root CA).

   2. Recovery from a compromised ACP registrar is difficult. When an
       ACP registrar is compromised, it can insert for example
       conflicting ACP domain information and create thereby an attack
       against other ACP nodes through the ACP routing protocol.

   Even when such a malicious ACP registrar is detected, resolving the
   problem may be difficult because it would require identifying all the
   wrong ACP domain certificates assigned via the ACP registrar after it
   was compromised. And without additional centralized tracking of
   assigned certificates there is no way to do this.

10.2.4. ACP Registrars with sub-CA

   In situations, where either of the above two limitations are an
   issue, ACP registrars could also be sub-CAs. This removes the need
   for connectivity to a root-CA whenever an ACP node is enrolled, and
   reduces the need for connectivity of such an ACP registrar to a root-
   CA to only those times when it needs to renew its own certificate.
   The ACP registrar would also now use its own (sub-CA) certificate to
   enroll and sign the ACP nodes certificates, and therefore it is only
   necessary to revoke a compromised ACP registrars sub-CA certificate.
   Or let it expire and not renew it, when the certificate of the sub-CA
   is appropriately short-lived.

   As the ACP domain membership check verifies a peer ACP node's ACP
   domain certificate trust chain, it will also verify the signing
   certificate which is the compromised/revoked sub-CA certificate.
   Therefore ACP domain membership for an ACP node enrolled from a
   compromised ACP registrar will fail.

ACP nodes enrolled by a compromised ACP registrar would automatically fail to establish ACP channels and ACP domain certificate renewal via EST and therefore revert to their role as a candidate ACP members and attempt to get a new ACP domain certificate from an ACP registrar - for example, via BRSKI. In result, ACP registrars that have an associated sub-CA makes isolating and resolving issues with compromised registrars easier.

Note that ACP registrars with sub-CA functionality also can control the lifetime of ACP domain certificates easier and therefore also be used as a tool to introduce short lived certificates and not rely on CRL, whereas the certificates for the sub-CAs themselves could be longer lived and subject to CRL.

10.2.5. Centralized Policy Control

When using multiple, uncoordinated ACP registrars, several advanced operations are potentially more complex than with a single, resilient policy control backend, for example including but not limited to:

   Which candidate ACP node is permitted or not permitted into an ACP domain. This may not be a decision to be taken upfront, so that a per-serialNumber policy can be loaded into ever ACP registrar. Instead, it may better be decided in real-time including potentially a human decision in a NOC.

   Tracking of all enrolled ACP nodes and their certificate information. For example in support of revoking individual ACP nodes certificates.

   More flexible policies what type of address prefix or even what specific address prefix to assign to a candidate ACP node.

These and other operations could be introduced more easily by introducing a centralized Policy Management System (PMS) and modifying ACP registrar behavior so that it queries the PMS for any policy decision occurring during the candidate ACP node enrollment process and/or the ACP node certificate renewal process. For example, which ACP address prefix to assign. Likewise the ACP registrar would report any relevant state change information to the PMS as well, for example when a certificate was successfully enrolled onto a candidate ACP node.

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, Section 10.3.2.2).

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 (autonomic) L2/L3 services running on the nodes. Therefore
one option is to change the operator action to not rely on link-state
propagation anymore. This may not be possible when both sides are
under different operator control, but in that case it is unlikely
that the ACP is running across the link and actually putting the
interface into "physical down" state may still be a good option.

Ideally, fast physical state propagation is replaced by fast software
driven state propagation. For example a DULL GRASP "admin-state"
objective could be used to auto configure a Bidirectional Forwarding
Protocol (BFD, [RFC5880]) session between the two sides of the link
that would be used to propagate the "up" vs. admin down state.

Triggering physical down state may also be used as a mean of
diagnosing cabling in the absence of easier methods. It is more
complex than automated neighbor diagnostics because it requires
coordinated remote access to both (likely) sides of a link to
determine whether up/down toggling will cause the same reaction on
the remote side.

See Section 10.1 for a discussion about how LLDP and/or diagnostics
via GRASP could be used to provide neighbor diagnostics, and
therefore hopefully eliminating the need for "physical down" for
neighbor diagnostics - as long as both neighbors support ACP/ANI.

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 auto
configuring links.

In addition to "Physical down", low level diagnostics of Ethernet or
other interfaces also involve the creation of other states on
interfaces, such as physical Loopback (internal and/or external) or
bringing down all packet transmissions for reflection/cable-length
measurements. Any of these options would disrupt ACP as well.

In cases where such low-level diagnostics of an operational link is
desired but where the link could be a single point of failure for the
ACP, ASA on both nodes of the link could perform a negotiated
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 Issues

Power consumption of "physical down" interfaces, may be significantly
lower than those in "admin down" state, for example on long-range
fiber interfaces. Bringing up interfaces, for example to probe
reachability, may also consume additional power. This can make these
type of interfaces inappropriate to operate purely for the ACP when
they are not currently needed for the Data-Plane.

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 port could
lead to immediate termination of BRSKI, while other parallel auto
configuration 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.

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. See Section 9.2 for details of how the ACP protects
itself against attacks from the outside and to a more limited degree
from the inside as well.

However, the security of the ACP depends on a number of other
factors:

o 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.

o 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
node's 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 Appendix A.6.

The ACP is designed to minimize attacks from the outside by
minimizing its dependency against any non-ACP (Data-Plane)
operations/configuration on a node. See also Section 6.12.2.

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 Appendix A.2.

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.

   "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 9) / 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, Elwyn Davies, Joel Halpern 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, review or suggestions were received from: Rene Struik,
   Brian Carpenter, Benoit Claise, William Atwood and Yongkang Zhang.

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

   o Clarified that the ACP should be based on, and support only IPv6.

   o Clarified in intro that ACP is for both, between devices, as well
      as for access from a central entity, such as an NMS.

   o Added a section on how to connect an NMS system.

   o Clarified the hop-by-hop crypto nature of the ACP.

   o Added several references to GDNP as a candidate protocol.

   o 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:

   o Introduced a new section "overview", to ease the understanding of
      the approach.

   o Merged the previous "problem statement" and "use case" sections
      into a mostly re-written "use cases" section, since they were
      overlapping.

   o Clarified the relationship with draft-ietf-anima-stable-
      connectivity

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

   o Took out requirement for IPv6 --> that's in the reference doc.

   o Added requirement section.

   o 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

      o Added some paragraphs in addressing section on "why IPv6 only", to
         reflect the discussion on the list.

      o 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.

      o 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.

      o 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.

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

      o Formalized 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)

      o 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.

      o 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.

      o Updated links, lots of small edits.

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

      o Added explicitly text for the ACP channel negotiation.

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

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

   o 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.

   o 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.

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

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

   o Deleted one sub-addressing scheme, focusing on a single scheme
      now.

   o Included information on how ANIMA information must be encoded in
      the domain certificate in section "preconditions".

   o 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

   o Section 5.3 (candidate ACP neighbor selection): Add that Intent
      can override only AFTER an initial default ACP establishment.

   o Section 6.10.1 (addressing): State that addresses in the ACP are
      permanent, and do not support temporary addresses as defined in
      RFC4941.

   o Modified Section 6.3 to point to the GRASP objective defined in
      draft-carpenter-anima-ani-objectives. (and added that reference)

   o Section 6.10.2: changed from MD5 for calculating the first 40-bits
      to SHA256; reason is MD5 should not be used any more.

   o Added address sub-scheme to the IANA section.

   o Made the routing section more prescriptive.

   o Clarified in Section 8.1.1 the ACP Connect port, and defined that
      term "ACP Connect".

   o Section 8.2: Added some thoughts (from mcr) on how traversing a L3
      cloud could be automated.

   o Added a CRL check in Section 6.7.

   o Added a note on the possibility of source-address spoofing into
      the security considerations section.

   o 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

   o Added proposed RPL profile.

   o detailed DTLS profile - DTLS with any additional negotiation/
      signaling channel.

   o Fixed up text for ACP/GRE encap. Removed text claiming its
      incompatible with non-GRE IPsec and detailed it.

   o Added text to suggest admin down interfaces should still run ACP.

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

   o Changed author association.

   o Improved ACP connect setion (after confusion about term came up in
      the stable connectivity draft review). Added picture, defined
      complete terminology.

o 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.

o Removed IANA request to allocate registry for GRASP/TLS. This
   would come with future draft (see above).

o Gave the name "ACP information field" to the field in the
   certificate carrying the ACP address and domain name.

o 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.

o 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.

o 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.

o 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.

   o 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.

   o 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.

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

   o 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 describe two options: one
   across configured tunnels (GRE, IPinIP etc) a more efficient one via
   directed DULL.

   Moved decription of BRSKI to appendix 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 too 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 auto configuration 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.

   o The stable connectivity draft was vaguely describing ACP connect
      behavior that is better standardized in this ACP draft.

   o 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 autonomically.

   o 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 sub-interfaces use manually configured prefixes NOT
      covered by the ACP ULA prefixes.

   o 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.

   o 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 modularly build software (e.g.,
      ASA or OAM for VNF) into ACP devices.

o 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.

o Added section about use of GRASP over ACP connect.

o 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).

o 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:

o ACP discovery inside ACP is bad *doh*!.

o Better CA trust and revocation sentences.

o More details about RPL behavior in ACP.

o 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):

o AN Domain Names are FQDNs.

o Fixed bit length of schemes, numerical writing of bits (00b/01b).

o 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), explained 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
   build 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 verbiage for ACP not to assign
   the virtual interface link-local address from the underlying
   interface. Added 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 details:

   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 unicast 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-attacks (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 datatracker and the PDF versions as well. Because rfcdiff also compares the .txt

version, this -11 version was created 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 detailing 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:

   o 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.

   o Replaced term "autonomic FOOBAR" with "acp FOOBAR" as wherever
      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 domain 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:

   o 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.

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

o 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.

o 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 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.

14.19. draft-ietf-anima-autonomic-control-plane-13

Swap author list (with permission).

6.1.1. Eliminate blank lines in definition by making it a picture (reformatting only).

6.10.3.1 New paragraph: Explained how nodes using Zone-ID != 0 need to use Zone-ID != 0 in GRASP so that we can avoid routing/forwarding of Zone-ID = 0 prefixes.

Rest of feedback from review of -12, see https://raw.githubusercontent.com/anima-wg/autonomic-control-plane/master/draft-ietf-anima-autonomic-control-plane/12-feedback-reply.txt

Review from Brian Carpenter:

   various: Autonomous -> autonomic(ally) in all remaining occurrences.

   various: changed "manual (configured)" to "explicitly (configured)"
   to not exclude the option of (SDN controller) automatic configuration
   (no humans involved).

   1. Fixed reference to section 9.

   2. Added definition of loopback interface == internal interface.
   After discus on WG mailing lists, including 6man.

   6.1.2 Defined CDP/OCSP and pointed to RFC5280 for them.

   6.1.3 Removed "EST-TLS", no objective value needed or beneficial,
   added explanation paragraph why.

   6.2 Added to adjacency table the interface that a neighbor is
   discovered on.

   6.3 Simplified CDDL syntax: Only one method per AN_ACP objective
   (because of locators). Example with two objectives in GRASP message.

   6.8.1 Added note about link-local GRASP multicast message to avoid
   confusion.

   8.1.4 Added RFC8028 as recommended on hosts to better support VRF-
   select with ACP.

   8.2.1 Rewrote and Simplified CDDL for configured remote peer and
   explanations. Removed pattern option for remote peer. Not important
   enough to be mandated.

   Review thread started by William Atwood:

   2. Refined definition of VRF (vs. MPLS/VPN, LISP, VRF-LITE).

   2. Refined definition of ACP (ACP includes ACP GRASP instance).

   2. Added explanation for "zones" to terminology section and into
   Zone Addressing Sub Scheme section, relating it to RFC4007 zones
   (from Brian Carpenter).

   4. Fixed text for ACP4 requirement (Clients of the ACP must not be
   tied to specific protocol.).

   5. Fixed step 4. with proposed text.

   6.1.1 Included suggested explanation for rsub semantics.

6.1.3 must->MUST for at least one EST server in ACP network to autonomically renew certs.

6.7.2 normative: AND MUST NOT (permit weaker crypto options.

6.7.1.1 also included text denying weaker IPsec profile options.

6.8.2 Fixed description how to build ACP GRASP virtual interfaces. Added text that ACP continues to exist in absence of ACP neighbors.

various: Make sure all "zone" words are used consistently.

6.10.2/various: fixed 40-bit RFC4193 ULA prefix in all examples to 89b714f3db (thanks MichaelR).

6.10.1 Removed comment about assigned ULA addressing. Decision not to use it now ancient history of WG decision making process, not worth nothing anymore in the RFC.

Review from Yongkang Zhang:

6.10.5 Fixed length of Node-Numbers in ACP Vlong Addressing Sub-Scheme.

14.20. draft-ietf-anima-autonomic-control-plane-14

Disclaimer: All new text introduced by this revision provides only additional explanations/ details based on received reviews and analysis by the authors. No changes to behavior already specified in prior revisions.

Joel Halpern, review part 3:

Define/explain "ACP registrar" in reply to Joel Halpern review part 3, resolving primarily 2 documentation issues::

1. Unclear how much ACP depends on BRSKI. ACP document was referring unqualified to registrars and Registrar-ID in the addressing section without explaining what a registrar is, leading to the assumption it must be a BRSKI Registrar.

2. Unclear how the ACP addresses in ACP domain certificates are assigned because the BRSKI document does not defines this, but refers to this ACP document.

Wrt. 1: ACP does NOT depend on BRSKI registrars, instead ANY appropriate automated or manual mechanism can be used to enroll ACP nodes with ACP domain certificates. This revision calls defines such

mechanisms the "ACP registrar" and defines requirements. this is
non-normative, because it does not define specific mechanisms that
need to be support. In ANI devices, ACP Registrars are BRSKI
Registrars. In non-ANI ACP networks, the registrar may simply be a
person using CLI/web-interfaces to provision domain certificates and
set the ACP address correctly in the ACP domain certificate.

Wrt. 2.: The BRSKI document does rightfully not define how the ACP
address assignment and creation of the ACP domain information field
has to work because this is independent of BRSKI and needs to follow
the same rules whatever protocol/mechanisms are used to implement an
ACP Registrar. Another set of protocols that could be used instead
of BRSKI is Netconf/Netconf-Call-Home, but such an alternative ACP
Registrar solution would need to be specified in its own document.

Additional text/sections had to be added to detail important
conditions so that automatic certificate maintenance for ACP nodes
(with BRSKI or other mechanisms) can be done in a way that as good as
possible maintains ACP address information of ACP nodes across the
nodes lifetime because that ACP address is intended as an identifier
of the ACP node.

Summary of sections added:

o 6.1.3.5/6.1.3.6 (normative): re-enrollment of ACP nodes after
   certificate expiry/failure in a way that allows to maintain as
   much as possible ACP address information.

o 6.10.7 (normative): defines "ACP Registrar" including requirements
   and how it can perform ACP address assignment.

o 10.3 (informative): details / examples about registrars to help
   implementers and operators understand easier how they operate, and
   provide suggestion of models that a likely very useful (sub-CA
   and/or centralized policy management).

o 10.4 (informative): Explains the need for the multiple address
   sub-spaces defined in response to discuss with Joel.

Other changes:

Updated references (RFC8366, RFC8368).

Introduced sub-section headings for 6.1.3 (certificate maintenance)
because section became too long with newly added sub-sections. Also
some small text fixups/remove of duplicate text.

Gen-ART review, Elwyn Davies:

[RFC Editor: how can i raise the issue of problematic cross references of terms in the terminology section - rendering is problematic.].

4. added explanation for ACP4 (finally).

6.1.1 Simplified text in bullet list explaining rfc822 encoding.

6.1.3 refined second paragraph defining remembering of previous EST server and explaining how to do this with BRSKI.

9.1 Added paragraph outlining the benefit of the sub-CA Registrar option for supporting partitioned networks.

Roughly 100 more nits/minor fixes throughout the document. See: https://raw.githubusercontent.com/anima-wg/autonomic-control-plane/master/draft-ietf-anima-autonomic-control-plane/13-elwynd-reply.txt

Joel Halpern, review part 2:

6.1.1: added note about "+ +" format in address field when acp-address and rsub are empty.

6.5.10 - clarified text about V bit in Vlong addressing scheme.

6.10.3/6.10.4 - moved the Z bit field up front (directly after base scheme) and indicated more explicitly Z is part of selecting of the sub-addressing scheme.

Refined text about reaching CRL Distribution Point, explain why address as indicator to use ACP.

Note from Brian Carpenter: RFC Editor note for section reference into GRASP.

IOT directorate review from Pascal Thubert:

Various Nits/typos.

TBD: Punted wish for mentioning RFC reference titles to RFC editor for now.

1. Added section 1.1 - applicability, discussing protocol choices re. applicability to constrained devices (or not). Added notion of TCP/TLS via CoAP/DTLS to section 10.4 in support of this.

2. Added in-band / out-of-band into terminology.

5. Referenced section 8.2 for remote ACP channel configuration.

6.3 made M_FLOOD periods RECOMMENDED (less guesswork)

6.7.x Clarified conditional nature of MUST for the profile details of
IPsec parameters (aka: only 6.7.3 defines actual MUST for nodes,
prior notions only define the requirements for IPsec profiles IF
IPsec is supported.

6.8.1 Moved discussion about IP multicast, IGP, RPL for GRASP into a
new subsection in the informative part (section 10) to tighten up
text in normative part.

6.10.1 added another reference to stable-connectivity for interop
with IPv4 management.

6.10.1 removed mentioning of ULA-Random, term was used in email
discus of ULA with L=1, but term actually not defined in rfc4193, so
mentioning it is just confusing/redundant. Also added note about the
random hash being defined in this document, not using SHA1 from
rfc4193.

6.11.1.1 added suggested text about mechanisms to further reduce
opportunities for loop during reconvergence (active signaling options
from RFC6550).

6.11.1.3 made mode 2 MUST and mode 2 MAY (RPL MOP - mode of
operations). Removes ambiguity.

6.12.5 Added recommendation for RFC4429 (optimistic DAD).

Nits from Benjamin Kaduk: dTLS -> DTLS:

Review from Joel Halpern:

1. swapped order of "purposes" for ACP to match order in section 3.

1. Added notion about manageability of ACP gong beyond RFC7575
(before discussion of stable connectivity).

2. Changed definition of Intent to be same as reference model
(policy language instead of API).

6.1.1 changed BNF specification so that a local-part without acp-
address (for future extensions) would not be rfcSELF.+rsub but
simpler rfcSELF+rsub. Added explanation why rsub is in local-part.

Tried to eliminate unnecessary references to VRF to minimize assumption how system is designed.

6.1.3 Explained how to make CDP reachable via ACP.

6.7.2 Made it clearer that constrained devices MUST support DTLS if they cannot support IPsec.

6.8.2.1 clarified first paragraph (TCP retransmissions lightweight).

6.11.1 fixed up RPL profile text - to remove "VRF". Text was also buggy. mentioned control plane, but it's a forwarding/silicon issue to have these header.

6.12.5 Clarified how link-local ACP channel address can be derived, and how not.

8.2.1 Fixed up text to distinguish between configuration and model describing parameters of the configuration (spec only provides parameter model).

Various Nits.

14.21. draft-ietf-anima-autonomic-control-plane-15

Only reshuffling and formatting changes, but wanted to allow reviewers later to easily compare -13 with -14, and these changes in -15 mess that up too much.

increased TOC depth to 4.

Separated and reordered section 10 into an operational and a background and futures section. The background and futures could also become appendices if the layout of appendices in RFC format wasn't so horrible that you really only want to avoid using them (all the way after a lot of text like references that stop most readers from proceeding any further).

14.22. draft-ietf-anima-autonomic-control-plane-16

Mirja Kuehlewind:

Tightened requirements for ACP related GRASP objective timers.

Better text to introduce/explains baseline and constrained ACP profiles.

   IANA guideline: MUST only accept extensible last allocation for
   address sub-scheme.

   Moved section 11 into appendix.

   Warren Kumari:

   Removed "global routing table", replaced with "Data-Plane routing
   (and forwarding) tables.

   added text to indicate how routing protocols do like to have data-
   plane dependencies.

   Changed power consumption section re. admin-down state. Power needed
   to bring up such interfaces make t inappropriate to probe. Need to
   think more about best suggests -> beyond scope.

   Replaced "console" with out-of-band... (console/management ethernet).

   Various nits.

   Joel Halpern:

   Fixed up domain information field ABNF to eliminate confusion that
   rsub is not an FQDN but only a prefix to routing-subdomain.

   Corrected certcheck to separate out cert verification into lifetime
   validity and proof of ownership of private key.

   Fixed pagination for "ACP as security and transport substrate for
   GRASP" picture.

14.23. draft-ietf-anima-autonomic-control-plane-17

   Review Alissa Cooper:

   Main discuss point fixed by untangling two specific node type cases:

   NOC nodes have ACP domain cert without acp-address field. Are ACP
   domain members, but cannot build ACP secure channels (just end-to-end
   or nay other authentications.

   ACP nodes may have other methods to assign ACP address than getting
   it through the cert. This is indicated through new value 0 for acp-
   address in certificate.

   Accordingly modified texts in ABNF/explanation and Cert-Check
   section.

Other:

Better separation of normative text and considerations for "future" work:

- Marked missing chapters as Informative. Reworded requirements section to indicate its informative nature, changed requirements to _MUST_/_SHOULD_ to indicate these are not RFC2119 requirements but that this requirements section is really just in place of a separate solutions requirements document (that ANIMA was not allowed to produce).

- removed ca. 20 instances of "futures" in normative part of document.

- moved important instances of "futures" into new section A.10 (last section of appendix). These serve as reminder of work discussed during WG but not able to finish specifying it.

Eliminated perception that "rsub" (routing subdomain) is only beneficial with future work. Example in A.7.

Added RFC-editor note re formatting of references to terms defined in terminology section.

Using now correct RFC 8174 boilerplate.

Clarified semantic and use of manual ACP sub-scheme. Not used in certificates, only assigned via traditional methods. Use for ACP-connect subnets or the like.

Corrected text about Data-Plane dependencies of ACP. Appropriate implementations can be fully data-plane independent (without more spec work) if not sharing link-local address with Data-Plane. 6.12.2 text updated to discuss those (MAC address), A.10.2 discusses options that would require new standards work.

Moved all text about Intent into A.8 to clearly mark it as futures.

Changed suggestion of future insecure ACP option to future "end-to-end-security-only" option.

Various textual fixes.

Gen-ART review by Elwyn Davies:

Some fixes also mentioned by Alissa.

Added reference for OT.

Fixed notion that secure channel is not only a security association.

>20 good textual fixes. Thanks!

Other:

Added picture requested by Pascal Thubert about Dual-NOC (A.10.4).

Moved RFC-editor request for better first RFC reference closer to the top of the document.

Fixed typo /126 -> 127 for prefix length with zone address scheme.

Overlooked early SecDir review from frank.xialiang@huawei.com:

most issues fixed through other review in -16. Added reference to self-protection section 9.2 into security considerations section.

14.24. draft-ietf-anima-autonomic-control-plane-18

Too many word/grammar mistakes in -17.

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.

[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-03
          (work in progress), July 2018.

[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>.

[RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener
          Discovery Version 2 (MLDv2) for IPv6", RFC 3810,
          DOI 10.17487/RFC3810, June 2004,
          <https://www.rfc-editor.org/info/rfc3810>.

   [RFC4191] Draves, R. and D. Thaler, "Default Router Preferences and
              More-Specific Routes", RFC 4191, DOI 10.17487/RFC4191,
              November 2005, <https://www.rfc-editor.org/info/rfc4191>.

   [RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast
              Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005,
              <https://www.rfc-editor.org/info/rfc4193>.

   [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing
              Architecture", RFC 4291, DOI 10.17487/RFC4291, February
              2006, <https://www.rfc-editor.org/info/rfc4291>.

   [RFC4301] Kent, S. and K. Seo, "Security Architecture for the
              Internet Protocol", RFC 4301, DOI 10.17487/RFC4301,
              December 2005, <https://www.rfc-editor.org/info/rfc4301>.

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

   [RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless
              Address Autoconfiguration", RFC 4862,
              DOI 10.17487/RFC4862, September 2007,
              <https://www.rfc-editor.org/info/rfc4862>.

   [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax
              Specifications: ABNF", STD 68, RFC 5234,
              DOI 10.17487/RFC5234, January 2008,
              <https://www.rfc-editor.org/info/rfc5234>.

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246,
              DOI 10.17487/RFC5246, August 2008,
              <https://www.rfc-editor.org/info/rfc5246>.

   [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S.,
              Housley, R., and W. Polk, "Internet X.509 Public Key
              Infrastructure Certificate and Certificate Revocation List
              (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008,
              <https://www.rfc-editor.org/info/rfc5280>.

   [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>.

   [RFC6553]   Hui, J. and JP. Vasseur, "The Routing Protocol for Low-
               Power and Lossy Networks (RPL) Option for Carrying RPL
               Information in Data-Plane Datagrams", RFC 6553,
               DOI 10.17487/RFC6553, March 2012,
               <https://www.rfc-editor.org/info/rfc6553>.

   [RFC7030]   Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed.,
               "Enrollment over Secure Transport", RFC 7030,
               DOI 10.17487/RFC7030, October 2013,
               <https://www.rfc-editor.org/info/rfc7030>.

   [RFC7296]   Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T.
               Kivinen, "Internet Key Exchange Protocol Version 2
               (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October
               2014, <https://www.rfc-editor.org/info/rfc7296>.

   [RFC7676]   Pignataro, C., Bonica, R., and S. Krishnan, "IPv6 Support
               for Generic Routing Encapsulation (GRE)", RFC 7676,
               DOI 10.17487/RFC7676, October 2015,
               <https://www.rfc-editor.org/info/rfc7676>.

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

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-16 (work in progress), June 2018.

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

   [I-D.ietf-anima-reference-model]
             Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L.,
             and J. Nobre, "A Reference Model for Autonomic
             Networking", draft-ietf-anima-reference-model-06 (work in
             progress), February 2018.

   [I-D.ietf-netconf-zerotouch]
             Watsen, K., Abrahamsson, M., and I. Farrer, "Zero Touch
             Provisioning for Networking Devices", draft-ietf-netconf-
             zerotouch-22 (work in progress), June 2018.

   [I-D.ietf-roll-applicability-template]
             Richardson, M., "ROLL Applicability Statement Template",
             draft-ietf-roll-applicability-template-09 (work in
             progress), May 2016.

   [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-23 (work in progress), May 2018.

   [IEEE-802.1X]
             IEEE SA-Standards Board, "IEEE Standard for Local and
             Metropolitan Area Networks: Port-Based Network Access
             Control", February 2010,
             <http://standards.ieee.org/findstds/
             standard/802.1X-2010.html>.

   [LLDP]   IEEE SA-Standards Board, "IEEE Standard for Local and
             Metropolitan Area Networks: Station and Media Access
             Control Connectivity Discovery", June 2016,
             <https://standards.ieee.org/findstds/
             standard/802.1AB-2016.html>.

   [MACSEC]   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>.

   [RFC4007]   Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and
               B. Zill, "IPv6 Scoped Address Architecture", RFC 4007,
               DOI 10.17487/RFC4007, March 2005,
               <https://www.rfc-editor.org/info/rfc4007>.

   [RFC4364]   Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private
               Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February
               2006, <https://www.rfc-editor.org/info/rfc4364>.

   [RFC4429]   Moore, N., "Optimistic Duplicate Address Detection (DAD)
               for IPv6", RFC 4429, DOI 10.17487/RFC4429, April 2006,
               <https://www.rfc-editor.org/info/rfc4429>.

   [RFC4541]   Christensen, M., Kimball, K., and F. Solensky,
               "Considerations for Internet Group Management Protocol
               (IGMP) and Multicast Listener Discovery (MLD) Snooping
               Switches", RFC 4541, DOI 10.17487/RFC4541, May 2006,
               <https://www.rfc-editor.org/info/rfc4541>.

   [RFC4604]   Holbrook, H., Cain, B., and B. Haberman, "Using Internet
               Group Management Protocol Version 3 (IGMPv3) and Multicast
               Listener Discovery Protocol Version 2 (MLDv2) for Source-
               Specific Multicast", RFC 4604, DOI 10.17487/RFC4604,
               August 2006, <https://www.rfc-editor.org/info/rfc4604>.

   [RFC4607] Holbrook, H. and B. Cain, "Source-Specific Multicast for
              IP", RFC 4607, DOI 10.17487/RFC4607, August 2006,
              <https://www.rfc-editor.org/info/rfc4607>.

   [RFC4610] Farinacci, D. and Y. Cai, "Anycast-RP Using Protocol
              Independent Multicast (PIM)", RFC 4610,
              DOI 10.17487/RFC4610, August 2006,
              <https://www.rfc-editor.org/info/rfc4610>.

   [RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy
              Extensions for Stateless Address Autoconfiguration in
              IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007,
              <https://www.rfc-editor.org/info/rfc4941>.

   [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>.

   [RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection
              (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010,
              <https://www.rfc-editor.org/info/rfc5880>.

   [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>.

   [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>.

   [RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The
              Locator/ID Separation Protocol (LISP)", RFC 6830,
              DOI 10.17487/RFC6830, January 2013,
              <https://www.rfc-editor.org/info/rfc6830>.

   [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>.

   [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language",
              RFC 7950, DOI 10.17487/RFC7950, August 2016,
              <https://www.rfc-editor.org/info/rfc7950>.

   [RFC8028] Baker, F. and B. Carpenter, "First-Hop Router Selection by
              Hosts in a Multi-Prefix Network", RFC 8028,
              DOI 10.17487/RFC8028, November 2016,
              <https://www.rfc-editor.org/info/rfc8028>.

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

   [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert,
              "A Voucher Artifact for Bootstrapping Protocols",
              RFC 8366, DOI 10.17487/RFC8366, May 2018,
              <https://www.rfc-editor.org/info/rfc8366>.

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

15.3. URIs

   [1] https://en.wikipedia.org/wiki/Operational_Technology

   [2] https://en.wikipedia.org/wiki/Single-root_input/
       output_virtualization

Appendix A. Background and Futures (Informative)

   The following sections discuss additional background information
   about aspects of the normative parts of this document or associated
   mechanisms such as BRSKI (such as why specific choices were made by
   the ACP) and they provide discussion about possible future variations
   of the ACP.

A.1. ACP Address Space Schemes

   This document defines the Zone, Vlong and Manual sub address schemes
   primarily to support address prefix assignment via distributed,
   potentially uncoordinated ACP registrars as defined in
   Section 6.10.7. This costs 48/46-bit identifier so that these ACP
   registrar can assign non-conflicting address prefixes. This design
   does not leave enough bits to simultaneously support a large number
   of nodes (Node-ID) plus a large prefix of local addresses for every
   node plus a large enough set of bits to identify a routing Zone. In
   result, Zone, Vlong 8/16 attempt to support all features, but in via
   separate prefixes.

In networks that always expect to rely on a centralized PMS as
described above (Section 10.2.5), the 48/46-bits for the Registrar-ID
could be saved. Such variations of the ACP addressing mechanisms
could be introduced through future work in different ways. If the
prefix rfcSELF in the ACP information field was changed, incompatible
ACP variations could be created where every design aspect of the ACP
could be changed. Including all addressing choices. If instead a
new addressing sub-type would be defined, it could be a backward
compatible extension of this ACP specification. Information such as
the size of a zone-prefix and the length of the prefix assigned to
the ACP node itself could be encoded via the extension field of the
ACP domain information.

Note that an explicitly defined "Manual" addressing sub-scheme is
always beneficial to provide an easy way for ACP nodes to prohibit
incorrect manual configuration of any non-"Manual" ACP address spaces
and therefore ensure that "Manual" operations will never impact
correct routing for any non-"Manual" ACP addresses assigned via ACP
domain certificates.

A.2. 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/DNS forwarding/server setup). This includes otherwise
not configured networks as described in Section 3.2. Therefore BRSKI
in conjunction with ACP provides for a secure and zero-touch
management solution for complete networks. Nodes supporting such an
infrastructure (BRSKI and ACP) are called ANI nodes (Autonomic
Networking Infrastructure), see [I-D.ietf-anima-reference-model].
Nodes that do not support an IDevID but only an (insecure) vendor
specific Unique Device Identifier (UDI) or nodes whose manufacturer
does not support a MASA could use some future security reduced
version of BRSKI.

When BRSKI is used to provision a domain certificate (which is called
enrollment), the BRSKI registrar (acting as an enhanced EST server)
must include the 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 "Do-not-renew" list would be
necessary on BRSKI registrars/CA.

In the absence of BRSKI or less secure variants thereof, provisioning
of certificates may involve one or more touches or non-standardized
automation. Node vendors usually support provisioning of
certificates into nodes via PKCS#7 (see [RFC2315]) and may support
this provisioning through vendor specific models via Netconf
([RFC6241]). If such nodes also support Netconf Zero-Touch
([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.

A.3. ACP Neighbor discovery protocol selection

This section discusses why GRASP DULL was chosen as the discovery
protocol for L2 adjacent candidate ACP neighbors. The contenders
considered where GRASP, mDNS or LLDP.

A.3.1. LLDP

LLDP and Cisco's earlier Cisco Discovery Protocol (CDP) are example
of L2 discovery protocols that terminate their messages on L2 ports.
If those protocols would be chosen for ACP neighbor discovery, ACP
neighbor discovery would therefore also terminate on L2 ports. This
would prevent ACP construction over non-ACP capable but LLDP or CDP
enabled L2 switches. LLDP has extensions using different MAC
addresses and this could have been an option for ACP discovery as
well, but the additional required IEEE standardization and definition
of a profile for such a modified instance of LLDP seemed to be more
work than the benefit of "reusing the existing protocol" LLDP for
this very simple purpose.

A.3.2. mDNS and L2 support

   Multicast DNNS (mDNS) [RFC6762] with DNS Service Discovery (DNS-SD)
   Resource Records (RRs) as defined in [RFC6763] is a key contender as
   an ACP discovery protocol. because it relies on link-local IP
   multicast, it does operates at the subnet level, and is also found in
   L2 switches. The authors of this document are not aware of mDNS
   implementation that terminate their mDNS messages on L2 ports instead
   of the subnet level. If mDNS was used as the ACP discovery mechanism
   on an ACP capable (L3)/L2 switch as outlined in Section 7, then this
   would be necessary to implement. It is likely that termination of
   mDNS messages could only be applied to all mDNS messages from such a
   port, which would then make it necessary to software forward any non-
   ACP related mDNS messages to maintain prior non-ACP mDNS
   functionality. Adding support for ACP into such L2 switches with
   mDNS could therefore create regression problems for prior mDNS
   functionality on those nodes. With low performance of software
   forwarding in many L2 switches, this could also make the ACP risky to
   support on such L2 switches.

A.3.3. Why DULL GRASP

   LLDP was not considered because of the above mentioned issues. mDNS
   was not selected because of the above L2 mDNS considerations and
   because of the following additional points:

   If mDNS was not already existing in a node, it would be more work to
   implement than DULL GRASP, and if an existing implementation of mDNS
   was used, it would likely be more code space than a separate
   implementation of DULL GRASP or a shared implementation of DULL GRASP
   and GRASP in the ACP.

A.4. Choice of routing protocol (RPL)

   This section motivates why RPL - "IPv6 Routing Protocol for Low-Power
   and Lossy Networks ([RFC6550] was chosen as the default (and in this
   specification only) routing protocol for the ACP. The choice and
   above explained profile was derived from a pre-standard
   implementation of ACP that was successfully deployed in operational
   networks.

   Requirements for routing in the ACP are:

   o 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]).

o 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.

o 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.

o 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.

o 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.

o 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.

o 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 optimization: RPL does not necessarily compute
   the optimal path between any two nodes. However, the ACP does not
   require this today, since it carries mainly non-delay-sensitive
   feedback loops. It is possible that different optimization
   schemes become necessary in the future, but RPL can be expanded
   (see point "Extensibility" above).

A.5. ACP Information Distribution and multicast

IP multicast is not used by the ACP because the ANI (Autonomic
Networking Infrastructure) itself does not require IP multicast but
only service announcement/discovery. Using IP multicast for that
would have made it necessary to develop a zero-touch auto configuring
solution for ASM (Any Source Multicast - the original form of IP
multicast defined in [RFC1112]), which would be quite complex and
difficult to justify. One aspect of complexity where no attempt at a
solution has been described in IETF documents is the automatic-
selection of routers that should be PIM Sparse Mode (PIM-SM)
Rendezvous Points (RPs) (see [RFC7761]). The other aspects of
complexity are the implementation of MLD ([RFC4604]), PIM-SM and
Anycast-RP (see [RFC4610]). If those implementations already exist
in a product, then they would be very likely tied to accelerated
forwarding which consumes hardware resources, and that in return is
difficult to justify as a cost of performing only service discovery.

Some future ASA may need high performance in-network data
replication. That is the case when the use of IP multicast is
justified. Such an ASA can then use service discovery from ACP
GRASP, and then they do not need ASM but only SSM (Source Specific
Multicast, see [RFC4607]) for the IP multicast replication. SSM
itself can simply be enabled in the Data-Plane (or even in an update
to the ACP) without any other configuration than just enabling it on
all nodes and only requires a simpler version of MLD (see [RFC5790]).

LSP (Link State Protocol) based IGP routing protocols typically have
a mechanism to flood information, and such a mechanism could be used
to flood GRASP objectives by defining them to be information of that
IGP. This would be a possible optimization in future variations of
the ACP that do use an LSP routing protocol. Note though that such a
mechanism would not work easily for GRASP M_DISCOVERY messages which
are intelligently (constrained) flooded not across the whole ACP, but
only up to a node where a responder is found. We do expect that many
future services in ASA will have only few consuming ASA, and for
those cases, M_DISCOVERY is the more efficient method than flooding
across the whole domain.

Because the ACP uses RPL, one desirable future extension is to use
RPLs existing notion of 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.

A.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 MacSec are to be added as negotiation
options - and the performance objective should work across all IPsec,
DTLS and MacSec 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:

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

o 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.

o 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.

A.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 permits the use of
   addresses from different ULA prefixes. One use case is to create
   multiple physical networks that initially may be separated with one
   ACP domain but different routing subdomains, so that all nodes can
   mutual trust their ACP domain certificates (not depending on rsub)
   and so that they could connect later together into a contiguous ACP
   network.

   One instance of such a use case is an ACP for regions interconnected
   via a non-ACP enabled core, for example due to the absence of product
   support for ACP on the core nodes. ACP connect configurations as
   defined in this document can be used to extend and interconnect those
   ACP islands to the NOC and merge them into a single ACP when later
   that product support gap is closed.

   Note that RPL scales very well. It is not necessary to use multiple
   routing subdomains to scale ACP domains in a way 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".

A.8. Intent for the ACP

   Intent is the architecture component of autonomic networks according
   to [I-D.ietf-anima-reference-model] that allows operators to issue
   policies to the network. In a simple instance, Intent could simply
   be policies flooded across ACP GRASP and interpreted on every ACP
   node.

   One concern for future definitions of Intent solutions is the problem
   of circular dependencies when expressing Intent policies about the
   ACP itself.

   For example, Intent could indicate the desire to build an ACP across
   all domains that have a common parent domain (without relying on the
   rsub/routing-subdomain solution defined in this document). For
   example ACP nodes with domain "example.com", nodes of "example.com",
   "access.example.com", "core.example.com" and "city.core.example.com"
   should all establish one single ACP.

   If each domain has its own source of Intent, then the Intent would
   simply have to allow adding the peer domains trust anchors (CA) and
   domain names to the ACP domain membership check (Section 6.1.2) so
   that nodes from those other domains are accepted as ACP peers.

   If this Intent was to be originated only from one domain, it could
   likely not be made to work because the other domains will not build
   any ACP connection amongst each other, whether they use the same or
   different CA due to the ACP domain membership check.

   If the domains use the same CA one could change the ACP setup to
   permit for the ACP to be established between two ACP nodes with
   different acp-domain-names, but only for the purpose of disseminating
   limited information, such as Intent, but not to set up full ACP
   connectivity, specifically not RPL routing and passing of arbitrary
   GRASP information. Unless the Intent policies permit this to happen
   across domain boundaries.

   This type of approach where the ACP first allows Intent to operate
   and only then sets up the rest of ACP connectivity based on Intent
   policy could also be used to enable Intent policies that would limit
   functionality across the ACP inside a domain, as long as no policy
   would disturb the distribution of Intent. For example to limit

reachability across the ACP to certain type of nodes or locations of
nodes.

A.9. Adopting ACP concepts for other environments

   The ACP as specified in this document is very explicit about the
   choice of options to allow interoperable implementations. The
   choices made may not be the best for all environments, but the
   concepts used by the ACP can be used to build derived solutions:

   The ACP specifies the use of ULA and deriving its prefix from the
   domain name so that no address allocation is required to deploy the
   ACP. The ACP will equally work not using ULA but any other /48 IPv6
   prefix. This prefix could simply be a configuration of the ACP
   registrars (for example when using BRSKI) to enroll the domain
   certificates - instead of the ACP registrar deriving the /48 ULA
   prefix from the AN domain name.

   Some solutions may already have an auto-addressing scheme, for
   example derived from existing unique device identifiers (e.g., MAC
   addresses). In those cases it may not be desirable to assign
   addresses to devices via the ACP address information field in the way
   described in this document. The certificate may simply serve to
   identify the ACP domain, and the address field could be 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].

TCP/TLS as the protocols to provide reliability and security to GRASP
in the ACP may not be the preferred choice in constrained networks.
For example, CoAP/DTLS (Constrained Application Protocol) may be
preferred where they are already used, allowing to reduce the
additional code space footprint for the ACP on those devices.
Because the transport for GRASP is not only hop-by-hop, but end-to-
end across the ACP, this would require the definition of an
incompatible variant of the ACP. Non-constrained devices could
support both variants (the ACP as defined here, and one using CoAP/
DTLS for GRASP), and the variant used in a deployment could be chosen
for example through a parameter of the domain certificate.

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 or introduce
more advanced RPL profiles.

Variations such as what routing protocol to use, or whether to
instantiate an ACP in a VRF or (as suggested above) as the actual
Data-Plane, can be automatically chosen in implementations built to
support multiple options by deriving them from future parameters in
the certificate. Parameters in certificates should be limited to
those that would not need to be changed more often than certificates
would need to be updated anyhow; Or by ensuring that these parameters
can be provisioned before the variation of an ACP is activated in a
node. Using BRSKI, this could be done for example as additional
follow-up signaling directly after the certificate enrollment, still
leveraging the BRSKI TLS connection and therefore not introducing any
additional connectivity requirements.

Last but not least, secure channel protocols including their
encapsulations are easily added to ACP solutions. ACP hop-by-hop
network layer secure channels could also be replaced by end-to-end
security plus other means for infrastructure protection. Any future
network OAM should always use end-to-end security anyhow and can
leverage the domain certificates and is therefore not dependent on
security to be provided for by ACP secure channels.

A.10. Further options / futures

A.10.1. Auto-aggregation of routes

Routing in the ACP according to this specification only leverages the
standard RPL mechanism of route optimization, e.g. keeping only
routes that are not towards the RPL root. This is known to scale to
networks with 20,000 or more nodes. There is no auto-aggregation of
routes for /48 ULA prefixes (when using rsub in the domain
information field) and/or Zone-ID based prefixes.

Automatic assignment of Zone-ID and auto-aggregation of routes could be achieved for example by configuring zone-boundaries, announcing via GRASP into the zones the zone parameters (zone-ID and /48 ULA prefix) and auto-aggegating routes on the zone-boundaries. Nodes would assign their Zone-ID and potentially even /48 prefix based on the GRASP announcements.

A.10.2. More options for avoiding IPv6 Data-Plane dependency

As described in Section 6.12.2, the ACP depends on the Data-Plane to establish IPv6 link-local addressing on interfaces. Using a separate MAC address for the ACP allows to fully isolate the ACP from the data plane in a way that is compatible with this specification. It is also an ideal option when using Single-root input/output virtualization (SR-IOV - see https://en.wikipedia.org/wiki/Single-root_input/output_virtualization [2]) in an implementation to isolate the ACP because different SR-IOV interfaces use different MAC addresses.

When additional MAC address(es) are not available, separation of the ACP could be done at different demux points. The same subnet interface could have a separate IPv6 interface for the ACP and Data-Plane and therefore separate link-local addresses for both, where the ACP interface is non-configurable on the Data-Plane. This too would be compatible with this specification and not impact interoperability.

An option that would require additional specification is to use a different Ethertype from 0x86DD (IPv6) to encapsulate IPv6 packets for the ACP. This would be a similar approach as used for IP authentication packets in [IEEE-802.1X] which use the Extensible Authentication Protocol over Local Area Network (EAPoL) ethertype (0x88A2).

Note that in the case of ANI nodes, all the above considerations equally apply to the encapsulation of BRSKI packets including GRASP used for BRSKI.

A.10.3. ACP APIs and operational models (YANG)

Future work should define YANG ([RFC7950]) data model and/or node internal APIs to monitor and manage the ACP.

Support for the ACP Adjacency Table (Section 6.2) and ACP GRASP need to be included into such model/API.

A.10.4. RPL enhancements

```
     ..... USA ......              ..... Europe ......

        NOC1                          NOC2
         |                             |
         |            metric 100       |
        ACP1 -------------------------- ACP2   .
         |                             |   . WAN
         | metric 10       metric 20   |   . Core
         |                             |   .
        ACP3 -------------------------- ACP4   .
         |            metric 100       |
         |                             |   .
         |                             |   . Sites
        ACP10                         ACP11  .
```
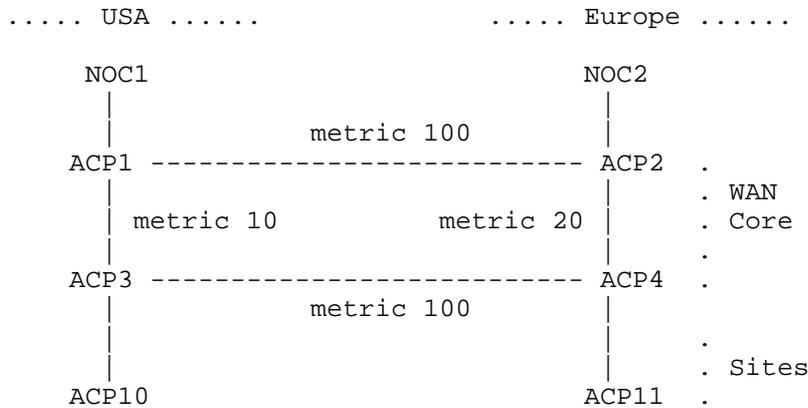
Figure 16: Dual NOC

The profile for RPL specified in this document builds only one
spanning-tree path set to a root (NOC). In the presence of multiple
NOCs, routing toward the non-root NOCs may be suboptimal. Figure 16
shows an extreme example. Assuming that node ACP1 becomes the RPL
root, traffic between ACP11 and NOC2 will pass through
ACP4-ACP3-ACP1-ACP2 instead of ACP4-ACP2 because the RPL calculated
DODAG/routes are shortest paths towards the RPL root.

To overcome these limitations, extensions/modifications to the RPL
profile can provide optimality for multiple NOCs. This requires
utilizing Data-Plane artifact including IPinIP encap/decap on ACP
routers and processing of IPv6 RPI headers. Alternatively, (Src,Dst)
routing table entries could be used.

Flooding of ACP GRASP messages can be further constrained and
therefore optimized by flooding only via links that are part of the
RPL DODAG.

A.10.5. Role assignments

ACP connect is an explicit mechanism to "leak" ACP traffic explicitly
(for example in a NOC). It is therefore also a possible security gap
when it is easy to enable ACP connect on arbitrary compromised ACP
nodes.

One simple solution is to define an extension in the ACP certificates
ACP information field indicating the permission for ACP connect to be

configured on that ACP node. This could similarly be done to decide whether a node is permitted to be a registrar or not.

Tying the permitted "roles" of an ACP node to the ACP domain certificate provides fairly strong protection against misconfiguration, but is still subject to code modifications.

Another interesting role to assign to certificates is that of a NOC node. This would allow to limit certain type of connections such as OAM TLS connections to only NOC initiator or responders.

A.10.6. Autonomic L3 transit

In this specification, the ACP can only establish autonomic connectivity across L2 hops and only explicitly configured options to tunnel across L3. Future work should specify mechanisms to automatically tunnel ACP across L3 networks. A hub&spoke option would allow to tunnel across the Internet to a cloud or central instance of the ACP, a peer-to-peer tunneling mechanism could tunnel ACP islands across an L3VPN infrastructure.

A.10.7. Diagnostics

Section 10.1 describes diagnostics options that can be done without changing the external, interoperability affecting characteristics of ACP implementations.

Even better diagnostics of ACP operations is possible with additional signaling extensions, such as:

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

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

3. The IDevID 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.

Authors' Addresses

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

   Email: tte+ietf@cs.fau.de

   Michael H. Behringer (editor)

   Email: michael.h.behringer@gmail.com

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

   Email: sbjarnason@arbor.net

ANIMA WG   M. Pritikin
Internet-Draft   Cisco
Intended status: Standards Track   M. Richardson
Expires: July 21, 2019   Sandelman
                                                     M. Behringer

                                                     S. Bjarnason
                                                  Arbor Networks
                                                       K. Watsen
                                                Juniper Networks
                                                January 17, 2019

        Bootstrapping Remote Secure Key Infrastructures (BRSKI)
              draft-ietf-anima-bootstrapping-keyinfra-18

Abstract

   This document specifies automated bootstrapping of an Autonomic
   Control Plane. To do this a remote secure key infrastructure (BRSKI)
   is created using manufacturer installed X.509 certificate, in
   combination with a manufacturer'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.

Copyright Notice

Table of Contents

1. Introduction

   BRSKI provides a solution for secure zero-touch (automated) bootstrap
   of virgin (untouched) devices that are called pledges in this
   document.

   This document primarily provides for the needs of the ISP and
   Enterprise focused ANIMA Autonomic Control Plane (ACP)
   [I-D.ietf-anima-autonomic-control-plane]. Other users of the BRSKI
   protocol will need to provide separate applicability statements that
   include privacy and security considerations appropriate to that
   deployment. Section Section 8 explains the details applicability for
   this the ACP usage.

This document describes how pledges discover (or be discovered by) an element of the network domain to which the pledge belongs to perform the bootstrap. This element (device) is called the registrar. Before any other operation, pledge and registrar need to establish mutual trust:

1. Registrar authenticating the pledge: "Who is this device? What is its identity?"

2. Registrar authorizing the pledge: "Is it mine? Do I want it? What are the chances it has been compromised?"

3. Pledge authenticating the registrar: "What is this registrar's identity?"

4. Pledge authorizing the registrar: "Should I join it?"

This document details protocols and messages to answer the above questions. It uses a TLS connection and an PKIX (X.509v3) certificate (an IEEE 802.1AR [IDevID] LDevID) of the pledge to answer points 1 and 2. It uses a new artifact called a "voucher" that the registrar receives from a "Manufacturer Authorized Signing Authority" and passes to the pledge to answer points 3 and 2.

A proxy provides very limited connectivity between the pledge and the registrar.

The syntactic details of vouchers are described in detail in [RFC8366]. 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) defined by [RFC8366].

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 that 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. Prior 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 (often pre-shared keys, including mechanisms like SIM cards)
   is pre-provisioned on each new device in a costly and non-scalable
   manner. Existing automated mechanisms are known as non-secured
   'Trust on First Use' (TOFU) [RFC7435], 'resurrecting duckling'
   [Stajano99theresurrecting] or 'pre-staging'.

   Another prior approach has been to try and minimize user actions
   during bootstrapping, but not eliminate all user-actions. The
   original EST protocol [RFC7030] does reduce user actions during
   bootstrap but does not provide solutions for how the following
   protocol steps can be made autonomic (not involving user actions):

   o using the Implicit Trust Anchor database to authenticate an owner
      specific service (not an autonomic solution because the URL must
      be securely distributed),

   o engaging a human user to authorize the CA certificate using out-
      of-band data (not an autonomic solution because the human user is
      involved),

   o using a configured Explicit TA database (not an autonomic solution
      because the distribution of an explicit TA database is not
      autonomic),

   o 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 manufacturer 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:

   o the pledge requires realtime connectivity to the manufacturer
      service,

   o the domain identity is exposed to the manufacturer service (this
      is a privacy concern),

   o the manufacturer 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 pinned-domain-cert leaf,
      i.e. the Registrars' certificate. 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 at
      the factory.

Voucher: A signed artifact from the MASA 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
   is asserted. Multiple voucher types are defined in [RFC8366]

Domain: The set of entities that share a common local 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 manages
   the private key 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". Within [I-D.ietf-anima-reference-model] this is
   refered to as the "join registrar autonomic service agent". Other
   communities use the abbreviation "JRC".

(Public) Key Infrastructure: The collection of systems and processes
   that sustain the activities of a public key system. The registrar
   acts as an [RFC5280] and [RFC5272] (see section 7) "Registration
   Authority".

Join Proxy: A domain entity that helps the pledge join the domain.
   A join 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. For simplicity this document sometimes uses the term of
   'proxy' to indicate the join proxy. The pledge is unaware that
   they are communicating with a proxy rather than directly with a
   registrar.

Circuit Proxy: A stateful implementation of the join proxy. This is
   the assumed type of proxy.

IPIP Proxy: A stateless proxy alternative.

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 [RFC8366]

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.

nonced: a voucher (or request) that contains a nonce (the normal
case).

nonceless: a voucher (or request) that does not contain a nonce,
relying upon accurate clocks for expiration, or which does not
expire.

manufacturer: the term manufacturer is used throughout this document
to be the entity that created the device. This is typically the
"original equipment manufacturer" or OEM, but in more complex
situations it could be a "value added retailer" (VAR), or possibly
even a systems integrator. In general, it a goal of BRSKI to
eliminate small distinctions between different sales channels.
The reason for this is that it permits a single device, with a
uniform firmware load, to be shipped directly to all customers.
This eliminates costs for the manufacturer. This also reduces the
number of products supported in the field increasing the chance
that firmware will be more up to date.

ANI: The Autonomic Network Infrastructure as defined by
[I-D.ietf-anima-reference-model]. This document details specific
requirements for pledges, proxies and registrars when they are
part of an ANI.

offline: When an architectural component cannot perform realtime
communications with a peer, either due to network connectivity or
because the peer is turned off, the operation is said to be
occurring offline.

1.3. Scope of solution

1.3.1. Support environment

   This solution (BRSKI) can support large router platforms with multi-
   gigabit inter-connections, mounted in controlled access data centers.
   But this solution is not exclusive to large equipment: 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 a 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 its
   presence through broadcasting.

   Nomadic or mobile devices often need to aquire credentials to access
   the network at the new location. An example of this is mobile phone
   roaming among network operators, or even between cell towers. This
   is usually called handoff. BRSKI does not provide a low-latency
   handoff which is usually a requirement in such situations. For these
   solutions BRSKI can be used to create a relationship (an LDevID) with
   the "home" domain owner. The resulting credentials are then used to
   provide credentials more appropriate for a low-latency handoff.

1.3.2. Constrained environments

   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).

   Specifically, there are protocol aspects described here that 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 that could be inappropriate.

1.3.3. Network Access Controls

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.

1.3.4. Bootstrapping is not Booting

This document describes "bootstrapping" as the protocol used to
obtain a local trust anchor. It is expected that this trust anchor,
along with any additional configuration information subsequently
installed, is persisted on the device across system restarts
("booting"). Bootstrapping occurs only infrequently such as when a
device is transfered to a new owner or has been reset to factory
default settings.

1.4. Leveraging the new key infrastructure / next steps

As a result of the protocol described herein, the bootstrapped
devices have the Domain CA trust anchor in common. An end entity
certificate has optionally been issued from the Domain CA. This
makes it possible to automatically deploy services across the domain
in a secure manner.

Services that benefit from this:

o Device management.

o Routing authentication.

o 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]).

1.5. Requirements for Autonomic Network Infrastructure (ANI) devices

The BRSKI protocol can be used in a number of environments. Some of
the flexibility in this document is the result of users out of the
ANI scope. This section defines the base requirements for ANI
devices.

For devices that intend to become part of an Autonomic Network
Infrastructure (ANI) ([I-D.ietf-anima-reference-model]) that includes
an Autonomic Control Plane
([I-D.ietf-anima-autonomic-control-plane]), the following actions are
required and MUST be performed by the pledge:

o BRSKI: Request Voucher

o EST: CA Certificates Request

o EST: CSR Attributes

o EST: Client Certificate Request

o BRSKI: Enrollment status Telemetry

The ANI Join Registrar ASA MUST support all the BRSKI and above
listed EST operations.

All ANI devices SHOULD support the BRSKI proxy function, using
circuit proxies. Other proxy methods are optional, and MUST NOT
enabled unless the Join Registrar ASA indicates support for them in
it's announcement. (See Section 4.3)

2. Architectural Overview

The logical elements of the bootstrapping framework are described in
this section. Figure 1 provides a simplified overview of the
components.

```
                                    +------------------------+
     +---------------Drop Ship--------------->| Vendor Service         |
     |                                        +------------------------+
     |                                        | M anufacturer|         |
     |                                        | A uthorized  |Ownership|
     |                                        | S igning     |Tracker  |
     |                                        | A uthority   |         |
     |                                        +--------------+---------+
     |                                                       ^
     |                                                       |  BRSKI-
     V                                                       |  MASA
   +-------+      ...............................................
   |       |      .                                          |   .
   |       |      .  +-----------+        +----------+       |   .
   |       |      .  |           |        |          |       |   .
   |Pledge |      .  |   Join    |        |  Domain  <-------+   .
   |       |      .  |   Proxy   |        | Registrar|           .
   |       <-------->............<-------> (PKI RA)  |           .
   |       |      .  |           |        |          |           .
   |       |      .  |           |  BRSKI-EST        |           .
   |IDevID |      .  +-----------+        +-----+-----+           .
   |       |      .                       | EST RFC7030           .
   |       |      .  +----------------+----------+               .
   |       |      .  | Key Infrastructure        |               .
   +-------+      .  | (e.g., PKI Certificate    |               .
                  .  |        Authority)         |               .
                  .  +---------------------------+               .
                  .                                              .
                  ...............................................
                              "Domain" components
```
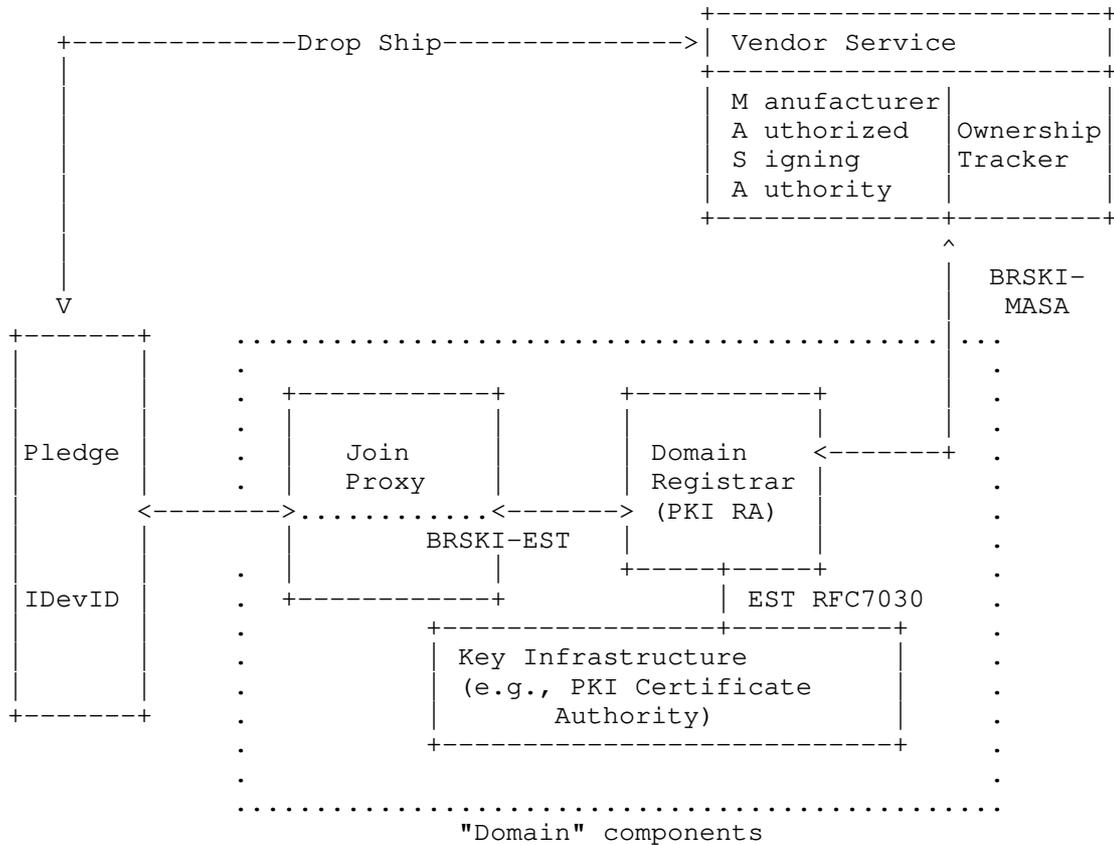
Figure 1

We assume a multi-vendor network. In such an environment there could
be a Manufacturer Service for each manufacturer that supports devices
following this document's specification, or an integrator could
provide a generic service authorized by multiple manufacturers. It
is unlikely that an integrator could provide Ownership Tracking
services for multiple manufacturers 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 domain provides initial
device connectivity sufficient for bootstrapping through a proxy.
The domain registrar authenticates the pledge, makes authorization
decisions, and distributes vouchers obtained from the Manufacturer
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.

```
                        +--------------+
                        │   Factory    │
                        │   default    │
                        +------+-------+
                               │
                        +------v-------+
                        │ (1) Discover │
     +----------->      │              │
     │                  +------+-------+
     │                         │
     │                  +------v-------+
     │                  │ (2) Identity │
     ^-----------+      │              │
     │ rejected  +------+-------+
     │                         │
     │                  +------v-------+
     │                  │ (3) Request  │
     │                  │     Join     │
     │                  +------+-------+
     │                         │
     │                  +------v-------+
     │                  │ (4) Imprint  │
     ^-----------+      │              │
     │ Bad MASA  +------+-------+
     │ response         │   send Voucher Status Telemetry
     │                  +------v-------+
     │                  │ (5) Enroll   │
     ^-----------+      │              │
     │ Enroll    +------+-------+
     │ Failure          │
     │                  +------v-------+
     │                  │ (6) Enrolled │
     ^-----------+      │              │
      Factory    +--------------+
      reset
```
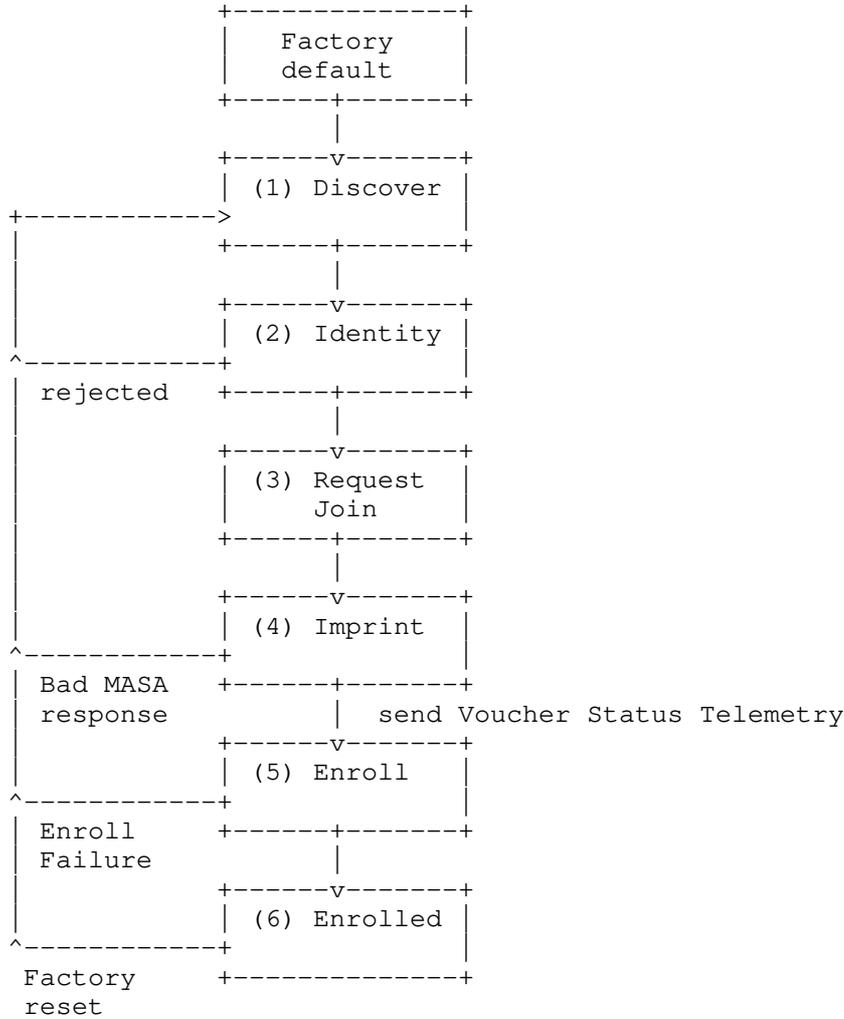
   Figure 2

   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. Request 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
    manufacturer service provided voucher. A voucher contains
    sufficient information for the pledge to complete authentication
    of a registrar. (The embedded 'pinned-domain-certificate'
    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.

After imprint a secure transport exists between pledge and registrar.
This specification details integration with EST enrollment so that
pledges can optionally obtain a locally issued certificate, although
any REST interface could be integrated in future work.

2.2. Secure Imprinting using Vouchers

A voucher is a cryptographically protected artifact (a digital
signature) 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 [RFC8366].

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 can indiscriminately issue vouchers
and log claims of ownership by domains. At the highest security
levels issuance of vouchers can be integrated with complex sales
channel integrations that are beyond the scope of this document. The
sales channel integration would verify actual (legal) ownership of
the pledge by the domain. 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 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
among 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 a
PKIX certificate installed during the manufacturing process. This is
the 802.1AR Initial Device Identifier (IDevID), and it provides a
basis for authenticating the pledge during the protocol exchanges
described here. There is no requirement for a common root PKI
hierarchy. Each device manufacturer can generate its own root
certificate. Specifically, the IDevID:

1. Uniquely identifying the pledge by the Distinguished Name (DN)
    and subjectAltName (SAN) parameters in the IDevID. The unique
    identification of a pledge in the voucher objects are derived
    from those parameters as described below.

2. Securely authenticating the pledges identity via TLS connection to
    registrar. This provides protection against cloned/fake pledged.

3. Secure auto-discovery of the pledges MASA by the registrar via
    the MASA URI in IDevID as explained below.

4. (Optionally) communicating the MUD URL (see Appendix C.

5. (Optional) Signing of voucher-request by the pledges IDevID to
    enable MASA to generate voucher only to a registrar that has a
    connection to the pledge.

6. Authorizing pledge (via registrar) to receive certificate from
    domain CA, by signing the Certificate Signing Request (CSR).

2.3.1. Identification of the Pledge

In the context of BRSKI, pledges are uniquely identified by a
"serial-number". This serial-number is used both in the "serial-
number" field of voucher or voucher-requests (see Section 3) and in
local policies on registrar or MASA (see Section 5).

The following fields are defined in [IDevID] and [RFC5280]:

o The subject field's DN encoding MUST include the "serialNumber"
   attribute with the device's unique serial number. (from [IDevID]
   section 7.2.8, and [RFC5280] section 4.1.2.4's list of standard
   attributes)

o The subject-alt field's encoding MAY include a non-critical
   version of the RFC4108 defined HardwareModuleName. (from [IDevID]
   section 7.2.9) If the IDevID is stored in a Trusted Platform
   Module (TPM), then this field MAY contain the TPM identification
   rather than the device's serial number. If both fields are
   present, then the subject field takes precedence.

and they are used as follows by the pledge to build the "serial-
number" that is placed in the voucher-request. In order to build it,
the fields need to be converted into a serial-number of "type
string". The following methods are used depending on the first
available IDevID certificate field (attempted in this order):

1. [RFC4519] section 2.31 provides an example ("WI-3005") of the
    Distinguished Name "serialNumber" attribute. [RFC4514] indicates
    this is a printable string so no encoding is necessary.

2. The HardwareModuleName hwSerialNum OCTET STRING. This value is
    base64 encoded to convert it to a printable string format.

The above process to locate the serial-number MUST be performed by
the pledge when filling out the voucher-request. Signed voucher-
requests are always passed up to the MASA, and the connection between
the serial-number in the voucher-request and the serial number in the
IDevID certificate.

As explained in Section 5.5 the Registrar MUST extract the serial-
number again itself from the pledge's TLS certificate. It may
consult the serial-number in the pledge-request if there are any
possible confusion about the source of the serial-number (hwSerialNum
vs serialNumber).

2.3.2. MASA URI extension

The following newly defined field SHOULD be in the PKIX IDevID
certificate: A PKIX 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 ([RFC5785]) 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:

```
     +--------+           +---------+         +-----------+    +------------+
     | Pledge |           | Circuit |         | Domain    |    | Vendor     |
     |        |           | Join    |         | Registrar |    | Service    |
     |        |           | Proxy   |         | (JRC)     |    | (MASA)     |
     +--------+           +---------+         +-----------+    +------------+
       |                    |                   |                    Internet |
       | <-RFC4862 IPv6 addr |                   |                           |
       | <-RFC3927 IPv4 addr | Appendix A        |   Legend                  |
       | ------------------->|                   |   C - circuit             |
       |   optional: mDNS query| Appendix B      |       join proxy          |
       |   RFC6763/RFC6762   |                   |   P - provisional         |
       | <------------------ |                   |       TLS connection      |
       |   GRASP M_FLOOD     |                   |                           |
       |      periodic broadcast|                |                           |
       | ------------------->C<----------------->|                           |
       |                 TLS via the Join Proxy  |                           |
       | <--Registrar TLS server authentication---|                          |
     [PROVISIONAL accept of server cert]         |                           |
       P---X.509 client authentication---------->|                           |
       P                    |                   |                           |
       P---Voucher Request (include nonce)------>|                           |
       P                    |            /--->  |                           |
       P                    |           |    [accept device?]              |
       P                    |           |    [contact Vendor]              |
       P                    |           |      |--Pledge ID-------->        |
       P                    |           |      |--Domain ID-------->        |
       P                    |           |      |--optional:nonce--->|       |
       P                    |           |           [extract DomainID]     |
       P                    |   optional:|          [update audit log]     |
       P                    |           |can   |                           |
       P                    |           |occur |                           |
       P                    |           |in    |                           |
       P                    |           |advance|                          |
       P                    |           |if    |                           |
       P                    |           |nonceless|                        |
       P                    |           |      |<- voucher ---------       |
       P                    |            \---->|                           |
       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

2.5. Architectural Components

2.5.1. Pledge

   The pledge is the device that is attempting to join. Until the
   pledge completes the enrollment process, it has link-local network
   connectivity only to the proxy.

2.5.2. Join Proxy

   The join proxy provides HTTPS connectivity between the pledge and the
   registrar. A circuit proxy mechanism is described in Section 4.
   Additional mechanisms, including a CoAP mechanism and a stateless
   IPIP mechanism are the subject of future work.

2.5.3. Domain Registrar

   The domain's registrar operates as the BRSKI-MASA client when
   requesting vouchers from the MASA (see Section 5.4). The registrar
   operates as the BRSKI-EST server when pledges request vouchers (see
   Section 5.1). The registrar operates as the BRSKI-EST server
   "Registration Authority" if the pledge requests an end entity
   certificate over the BRSKI-EST connection (see Section 5.9).

   The registrar uses an Implicit Trust Anchor database for
   authenticating the BRSKI-MASA TLS connection MASA certificate. The
   registrar uses a different Implicit Trust Anchor database for
   authenticating the BRSKI-EST TLS connection pledge client
   certificate. Configuration or distribution of these trust anchor
   databases is out-of-scope of this specification.

2.5.4. Manufacturer Service

   The Manufacturer Service provides two logically seperate functions:
   the Manufacturer Authorized Signing Authority (MASA) described in
   Section 5.5 and Section 5.6, and an ownership tracking/auditing
   function described in Section 5.7 and Section 5.8.

2.5.5. Public Key Infrastructure (PKI)

   The Public Key Infrastructure (PKI) administers certificates for the
   domain of concerns, providing the trust anchor(s) for it and allowing
   enrollment of pledges with domain certificates.

   The voucher provides a method for the distribution of a single PKI
   trust anchor (as the "pinned-domain-cert"). A distribution of the
   full set of current trust anchors is possible using the optional EST
   integration.

   The domain's registrar acts as an [RFC5272] Registration Authority,
   requesting certificates for pledges from the Key Infrastructure.

   The expectations of the PKI are unchanged from EST [[RFC7030]]. This
   document does not place any additional architectural requirements on
   the Public Key Infrastructure.

2.6. Certificate Time Validation

2.6.1. Lack of realtime clock

   Many devices when bootstrapping do not have knowledge of the current
   time. Mechanisms such as Network Time Protocols cannot 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:

   o Initially the pledge does not know the current time.

   o Bootstrapping pledges that have a Realtime Clock (RTC), SHOULD use
      it to verify certificate validity. However, they MUST be prepared
      for the recognize that the RTC might be completely wrong when a
      RTC battery fails and resets to an origin time (e.g., Jan. 1,
      1970)

   o If the pledge has any stable storage (such as from where firmware
      is loaded) then it SHOULD assume that the clock CAN NOT be before
      the date at which the firmware or the the storage was last time
      stamped. The pledge SHOULD NOT update the timestamps in any file
      systems until it has a secure time source. This provides an
      earliest date which is reasonable. Call this the current
      reasonable date (CRD). This value MUST NOT be used for any future
      Registration attempt. The current reasonable date (CRD) may only
      increase during a single attempt.

   o The pledge is exposed to dates in the following five places
      (registrar certificate, notBefore and notAfter. Voucher created-
      on, and expires-on. Additionally, CMS signatures contain a
      signingTime)

   o During the initial connection with the registrar, the pledge sees
      the registrar's Certificate. It has an inception date (notBefore)

and an expiry date (notAfter). It is reasonable that the
notBefore date be after the pledge's current working reasonable
date. It is however, suspicious for the notAfter date to be
before the pledge's current reasonable date. No action is
recommended, other than an internal audit entry for this.

o If the notBefore date of the registrar's certificate is newer than
   the pledge's reasonable date, then it MAY update it's current
   reasonable date to the notBefore value.

o After the voucher request process, the pledge will have a voucher.
   It can validate the signature on the voucher, as it has been (by
   literal construction) provided with the MASA's key as a trust
   anchor. The time values (created-on, expires-on) in the voucher
   can not in general be validated as the pledge has no certain real
   time clock. There are some reasonable assumptions that can be
   made: the voucher's expires-on time can not be prior to the
   pledge's current reasonable date. For nonceless vouchers, the
   voucher's created-on time COULD be earlier if the as well if a
   long-lived voucher was obtained some time in the past, and the
   pledge has since gone through a firmware update and factory reset.

o 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. In that case, the
   voucher's created-on date MUST NOT be prior to the pledge's
   current reasonable date. In addition, when there is a valid
   nonce, the current reasonable date MAY be incremented to that of
   the CMS signingTime.

o Once the voucher is accepted the validity period of the pinned-
   domain-cert in the voucher now serves as a valid time window. As
   explained in Section 5.5.4, the MASA has checked the registrar's
   certificate against real clocks , the endorsement of the MASA
   allows the pledge to treat the notBefore and notAfter dates as
   being constraints on any subsequent certificate validity periods
   that may need to be checked: for instance, validating peer
   certificates during ANIMA ACP setup.

o 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.

2.6.2. Infinite Lifetime of IDevID

   [RFC5280] explains that long lived pledge certificates "SHOULD be
   assigned the GeneralizedTime value of 99991231235959Z". Registrars
   MUST support such lifetimes and SHOULD support ignoring pledge
   lifetimes if they did not follow the RFC5280 recommendations.

   For example, IDevID may have incorrect lifetime of N <= 3 years,
   rendering replacement pledges from storage useless after N years
   unless registrars support ignoring such a lifetime.

2.7. Cloud Registrar

   There exist operationally open network wherein devices gain
   unauthenticated access to the internet at large. In these use cases
   the management domain for the device needs to be discovered within
   the larger internet. These are less likely within the anima scope
   but may be more important in the future.

   There are additionally some greenfield situations involving an
   entirely new installation where a device may have some kind of
   management uplink that it can use (such as via 3G network for
   instance). In such a future situation, the device might use this
   management interface to learn that it should configure itself by to-
   be-determined mechanism (such as an Intent) to become the local
   registrar.

   In order to support these scenarios, the pledge MAY contact a well
   known URI of a cloud registrar if a local registrar cannot 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.

2.8. 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 C.

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 manufacturer.
Note that the registrar can only select the configured MASA URL based
on the trust anchor -- so manufacturers 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

Voucher-requests are how vouchers are requested. The semantics of
the vouchers are described below, in the YANG model.

A pledge forms the "pledge voucher-request" and submits it to the
registrar.

The registrar in turn forms the "registrar voucher-request", and
submits it to the MASA.

The "proximity-registrar-cert" leaf is used in the pledge voucher-
requests. This provides a method for the pledge to assert the
registrar's proximity.

The "prior-signed-voucher-request" leaf is used in registrar voucher-
requests. If present, it is the encoded (signed form) of the pledge
voucher-request. This provides a method for the registrar to forward
the pledge's signed request to the MASA. This completes transmission
of the signed "proximity-registrar-cert" leaf.

A registrar MAY also retrieve nonceless vouchers by sending nonceless
voucher-requests to the MASA in order to obtain vouchers for use when
the registrar does not have connectivity to the MASA. No "prior-
signed-voucher-request" leaf would be included. The registrar will
also need to know the serial number of the pledge. This document
does not provide a mechanism for the registrar to learn that in an
automated fashion. Typically this will be done via scanning of bar-
code or QR-code on packaging, or via some sales channel integration.

Unless otherwise signaled (outside the voucher-request artifact), the
signing structure is as defined for vouchers, see [RFC8366].

3.1. Tree Diagram

   The following tree diagram illustrates a high-level view of a
   voucher-request document. The voucher-request builds upon the
   voucher artifact described in [RFC8366]. The tree diagram is
   described in [RFC8340]. 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-request examples for illustration
   purposes. 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). The pledge's voucher-request, if a signed artifact with a CMS format signature is a binary object. In the JSON encoding used here it must be base64 encoded. The nonce, created-on and assertion is carried forward. The serial-number is extracted from the pledge's Client Certificate from the TLS connection. See Section 5.5.

```
{
    "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 can not communicate with the registrar (such as when it is powered down, or still in packaging), and therefore could not submit a nonce. This scenario is most useful when the registrar is aware that it will not be able to reach the MASA during deployment. See Section 5.5.

```
{
    "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 its 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 5.5.

```
   {
       "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 [RFC8366]
   voucher into a voucher-request.

```
<CODE BEGINS> file "ietf-voucher-request@2018-02-14.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 "This module defines the format for a voucher,
        which is produced by a pledge's manufacturer or
        delegate (MASA) to securely assign a pledge to
        an 'owner', so that the pledge may establish a secure
        conn ection to the owner's network infrastructure";

    reference "RFC YYYY: 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 defines the format for a voucher request.
    It is a superset of the voucher itself.
    This artifact may be optionally signed.
    It provides content to the MASA for consideration
    during a voucher request.

    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 "2018-02-14" {
    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-request 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. Proxying details (Pledge - Proxy - Registrar)

   The role of the proxy is to facilitate communications. The proxy
   forwards packets between the pledge and a registrar that has been
   provisioned to the proxy via GRASP discovery.

   This section defines a stateful proxy mechanism which is refered to
   as a "circuit" 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.

   Registrars are assumed to have logically a locally integrated circuit
   proxy to support directly (subnet) connected pledges - because
   registrars themself does not define any functions for pledges to
   discover them. Such a logical local proxy does not need to provide
   actual TCP proxying (just discovery) as long as the registrar can
   operate with subnet (link) local addresses on the interfaces where
   pledges may connect to.

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

   In the ANI, the Autonomic Control Plane (ACP) secured instance of
   GRASP ([I-D.ietf-anima-grasp]) MUST be used for discovery of ANI
   registrar ACP addresses and ports by ANI proxies. The TCP leg of the
   proxy connection between ANI proxy and ANI registrar therefore also
   runs across the ACP.

   During the discovery of the Registrar by the Join Proxy, the Join
   Proxy will also learn which kinds of proxy mechanisms are available.
   This will allow the Join Proxy to use the lowest impact mechanism
   which the Join Proxy and Registrar have in common.

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.

While the GRASP M_FLOOD mechanism is passive for the pledge, the
optional other methods (mDNS, and IPv4 methods) are active. The
pledge SHOULD run those methods in parallel with listening to for the
M_FLOOD. The active methods SHOULD exponentially back-off to a
maximum of one hour to avoid overloading the network with discovery
attempts. Detection of change of physical link status (ethernet
carrier for instance) SHOULD reset the exponential back off.

The pledge could discover more than one proxy on a given physical
interface. The pledge can have a multitude of physical interfaces as
well: a layer-2/3 ethernet switch may have hundreds of physical
ports.

Each possible proxy offer SHOULD be attempted up to the point where a
voucher is received: while there are many ways in which the attempt
may fail, it does not succeed until the voucher has been validated.

The connection attempts via a single proxy SHOULD exponentially back-
off to a maximum of one hour to avoid overloading the network
infrastructure. The back-off timer for each MUST be independent of
other connection attempts.

Connection attempts SHOULD be run in parallel to avoid head of queue
problems wherein an attacker running a fake proxy or registrar could
perform protocol actions intentionally slowly. The pledge SHOULD
continue to listen to for additional GRASP M_FLOOD messages during
the connection attempts.

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 (assuming that none
succeeded) the device MUST return to listening for GRASP M_FLOOD. It
SHOULD periodically retry the manufacturer specific mechanisms. The
pledge MAY prioritize selection order as appropriate for the
anticipated environment.

4.1.1. Proxy GRASP announcements

A proxy uses the DULL GRASP M_FLOOD mechanism to announce itself.
This announcement can be within the same message as the ACP
announcement detailed in [I-D.ietf-anima-autonomic-control-plane].
The M_FLOOD is formatted as follows:

```
[M_FLOOD, 12340815, h'fe800000000000000000000000000001', 180000,
          ["AN_Proxy", 4, 1, ""],
          [O_IPv6_LOCATOR,
            h'fe800000000000000000000000000001', IPPROTO_TCP, 4443]]
```

Figure 6b: Proxy Discovery

The formal CDDL [I-D.ietf-cbor-cddl] definition is:

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

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

ttl             = 180000     ; 180,000 ms (3 minutes)
initiator = ACP address to contact Registrar
objective-flags   = sync-only  ; as in GRASP spec
sync-only         =  4         ; M_FLOOD only requires synchronization
loop-count        =  1         ; one hop only
objective-value   =  any       ; none

locator-option    = [ O_IPv6_LOCATOR, ipv6-address,
                     transport-proto, port-number ]
ipv6-address      = the v6 LL of the Proxy
$transport-proto /= IPPROTO_TCP   ; note this can be any value from the
                                  ; IANA protocol registry, as per
                                  ; [GRASP] section 2.9.5.1, note 3.
port-number       = selected by Proxy
```

   Figure 6c: AN_Proxy CDDL

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. 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 ACP instance of GRASP using
   M_FLOOD messages. They MUST support ANI TLS circuit proxy and
   therefore BRSKI across HTTPS/TLS native across the ACP. ANI proxies
   MUST support GRASP discovery of registrars.

   The M_FLOOD is formatted as follows:

```
[M_FLOOD, 12340815, h'fda379a6f6ee00000200000064000001', 180000,
          ["AN_join_registrar", 4, 255, "EST-TLS"],
          [O_IPv6_LOCATOR,
            h'fda379a6f6ee00000200000064000001', IPPROTO_TCP, 80]]
```

   Figure 7a: 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 sufficiently low that the aggregate amount of periodic M_FLOODs
from all EST servers causes negligible traffic across the ACP.

Here are some examples of locators for illustrative purposes. Only
the first one (transport-protocol = 6, TCP) is defined in this
document and is mandatory to implement.

```
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]
```

A protocol of 6 indicates that TCP proxying on the indicated port is
desired.

Registrars MUST announce the set of protocols that they support.
They MUST support TCP traffic.

Registrars MUST accept HTTPS/EST traffic on the TCP ports indicated.

5. Protocol Details (Pledge - Registrar - MASA)

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]. The goal of these
extensions is 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 existing CMS message formats for existing EST operations.
BRSKI 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 state
occurs only once, and that the subsequent resolution of the provision
state is not subject to a MITM attack during a critical phase.

Summarized automation extensions for the BRSKI-EST flow are:

o The pledge provisionally accepts the registrar certificate during
   the TLS handshake as detailed in Section 5.1.

o The pledge either attempts concurrent connections, or it times out
   quickly and tries connections in series.

o The pledge requests and validates a voucher using the new REST
   calls described below.

o The pledge completes authentication of the server certificate as
   detailed in Section 5.6.1. This moves the BRSKI-EST TLS
   connection out of the provisional state.

o Mandatory boostrap steps conclude with voucher status telemetry
   (see Section 5.7).

The BRSKI-EST TLS connection can now be used for EST enrollment.

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

o 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.

o 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 manufacturer trust
   anchor since serial numbers are not globally unique.

o The registrar requests and validates the voucher from the MASA.

o The registrar forwards the voucher to the pledge when requested.

o The registrar 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
   registrar. 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 an 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 5.6.1

   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 connect to any
   other registrar's via any other discovered proxies if there are any.
   If there were no other proxies discovered, the pledge MAY continue to
   wait, as long as it is concurrently listening for new proxy
   announcements.

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/voucher-cms+json The request is a "YANG-defined JSON
   document that has been signed using a CMS 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 the exception that it is not within a
   CMS 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 2.6.1 functionality. The nonce MUST NOT be reused for
   multiple 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 that the 'proximity' asserion and associated 'proximity-
registrar-cert' are correct.

5.3. Registrar Authorization of Pledge

   In a fully automated network all devices must be securely identified
   and authorized to join the domain.

   A Registrar accepts or declines a request to join the domain, based
   on the authenticated identity presented. Automated acceptance
   criteria include:

   o allow any device of a specific type (as determined by the X.509
      IDevID),

   o allow any device from a specific vendor (as determined by the
      X.509 IDevID),

   o allow a specific device from a vendor (as determined by the X.509
      IDevID) against a domain white list. (The mechanism for checking
      a shared white list potentially used by multiple Registrars is out
      of scope).

   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 5.5) and returns that MASA signed
   voucher to the pledge as described in Section 5.6.

5.4. 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 2.8
   for [RFC6125] authentication of the MASA.

   The primary method of registrar "authentication" by the MASA is
   detailed in Section 5.5. As detailed in Section 10 the MASA might
   find it necessary to request additional registrar authentication.

   The MASA and the registrars SHOULD be prepared to support TLS client
   certificate authentication and/or HTTP Basic or Digest authentication
   as described in RFC7030 for EST clients. This connection MAY also
   have no client authentication at all (Section 6.4)

   The authentication of the BRSKI-MASA connection does not affect the
   voucher-request process, as voucher-requests are already signed by
   the registrar. Instead, this authentication provides access control
   to the audit log.

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.

5.5. 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 via an HTTPS RESTful
   interface ([RFC7231]).

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

   The request media type is defined in [RFC8366] and is application/
   voucher-cms+json. It is a JSON document that has been signed using a
   CMS structure. 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 CMS 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. The registrar determines this value by
      parsing the authenticated pledge IDevID certificate. See
      Section 2.3. The registrar SHOULD verify that the serial number
      field it parsed matches the serial number field the pledge
      provided in its voucher-request. This provides a sanity check
      useful for detecting error conditions and logging. The registrar
      MUST NOT simply copy the serial number field from a pledge voucher
      request as that field is claimed but not certified.

   idevid-issuer: The idevid-issuer value from the pledge certificate
      is included to ensure a statistically unique identity.

   prior-signed-voucher-request: 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). If a signed
voucher-request was not recieved from the pledge then this leaf is
omitted from the registrar voucher request.

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 anticipates not being able to connect
to the MASA while the pledge is being deployed. Some 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 for 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 Examples 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 known to the MASA in advance.
The MASA performs the actions and validation checks described in the
following sub-sections before issuing a voucher.

5.5.1. MASA renewal of expired vouchers

As described in [RFC8366] vouchers are normally short lived to avoid
revocation issues. If the request is for a previous (expired)
voucher using the same registrar then the request for a renewed
voucher SHOULD be automatically authorized. The MASA has sufficient
information to determine this by examining the request, the registrar
authentication, and the existing audit log. The issuance of a
renewed voucher is logged as detailed in Section 5.6.

To inform the MASA that existing vouchers are not to be renewed one
can update or revoke the registrar credentials used to authorize the
request (see Section 5.5.3 and Section 5.5.4). More flexible methods
will likely involve sales channel integration and authorizations
(details are out-of-scope of this document).

5.5.2. MASA verification of voucher-request 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 the
voucher-request signer to be a registrar. Performing this check
provides value to the domain PKI by assuring the domain administrator
that the MASA service will only respect claims from authorized
Registration Authorities of the domain.

The MASA verifies that the domain CA certificate is included in the
CMS structure as detailed in Section 5.5.

5.5.3. MASA authentication of registrar (certificate)

If a nonceless voucher-request is submitted the MASA 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
(details are out-of-scope of this document).

In the nonced case, validation of the registrar MAY be omitted if the
device policy is to accept audit-only vouchers.

5.5.4. MASA revocation checking of registrar (certificate)

As noted in Section 5.5.3 the MASA performs registrar authentication
in a subset of situations (e.g. nonceless voucher requests). Normal
PKIX revocation checking is assumed during either EST client
authentication or voucher-request signature validation. Similarly,
as noted in Section 5.5.2, the MASA performs normal PKIX revocation
checking during signature consistency checks (a signature by a
registrar certificate that has been revoked is an inconsistency).

5.5.5. MASA verification of pledge prior-signed-voucher-request

The MASA MAY verify that the registrar voucher-request includes the
'prior-signed-voucher-request' field. If so the prior-signed-
voucher-request MUST include a 'proximity-registrar-cert' that is
consistent with the certificate used to sign the registrar voucher-
request. Additionally the voucher-request serial-number leaf MUST
match the pledge serial-number that the MASA extracts from the
signing certificate of the prior-signed-voucher-request. The MASA is
aware of which pledges support signing of their voucher requests and
can use this information to confirm proximity of the pledge with the
registrar, thus ensuring that the BRSKI-EST TLS connection has no
man-in-the-middle.

If these checks succeed the MASA updates the voucher and audit log
assertion leafs with the "proximity" assertion.

5.5.6. MASA pinning of registrar

   The registrar's certificate chain is extracted from the signature
   method. The chain includes the domain CA certificate as specified in
   Section 5.5. This 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.

5.5.7. MASA nonce handling

   The MASA does not verify the nonce itself. It MAY perform a simple
   consistency check: If the registrar voucher-request contains a nonce
   and the prior-signed-voucher-request exists then the nonce in both
   MUST be consistent. (Recall from above that the voucher-request
   might not contain a nonce, see Section 5.5 and Section 5.5.3).

   The MASA MUST use the nonce from the registrar voucher-request for
   the resulting voucher and audit log. The prior-signed-voucher-
   request nonce is ignored during this operation.

5.6. MASA and Registrar Voucher Response

   The MASA voucher response to the registrar is forwarded without
   changes to the pledge; therefore this section applies to both the
   MASA and the registrar. The HTTP signaling described applies to both
   the MASA and registrar responses. A registrar either caches prior
   MASA responses or dynamically requests a new voucher based on local
   policy (it does not generate or sign a voucher).

   If the voucher-request is successful, the server (MASA responding to
   registrar or registrar responding to pledge) 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. In this
   case, the response data from the MASA MUST be a plaintext human-
   readable (ASCII, English) error message containing explanatory
   information describing why the request was rejected.

   The registrar 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". (see [RFC7231] section 6.6.4) The pledge is
   RECOMMENDED to provide local feedback (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. Ideally the
   pledge would keep track of the appropriate Retry-After header values

for any number of outstanding registrars but this would involve a
state table on the pledge. Instead the pledge MAY ignore the exact
Retry-After value in favor of a single hard coded value. A registrar
that is unable to complete the transaction the first time due to
timing reasons will have future chances.

In order to avoid infinite redirect loops, which a malicious
registrar might do in order to keep the pledge from discovering the
correct registrar, the pledge MUST NOT follow more than one
redirection (3xx code) to another web origins. EST supports
redirection but requires user input; this change allows the pledge to
follow a single redirection without a user interaction.

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

A 404 (Not Found) response is appropriate when the request is for a
device that 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) cannot be
issued such as because the MASA knows the pledge cannot process that
type. The registrar SHOULD use this response if it determines the
pledge is unacceptable due to inventory control, MASA audit logs, or
any other reason.

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/voucher-cms+json The response is a "YANG-defined JSON
   document that has been signed using a CMS structure" as described
   in [RFC8366] using the JSON encoded described in [RFC7951]. The
   MASA MUST sign the response.

The syntactic details of vouchers are described in detail in
[RFC8366]. For example, the voucher consists of:

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

The MASA populates the voucher fields as follows:

nonce: The nonce from the pledge if available. See Section 5.5.7.

assertion: The method used to verify assertion. See Section 5.5.5.

pinned-domain-cert: The domain CA cert. See Section 5.5.6.

serial-number: The serial-number as provided in the voucher-request.
   Also see Section 5.5.5.

domain-cert-revocation-checks: Set as appropriate for the pledge's
   capabilities and as documented in [RFC8366]. The MASA MAY set
   this field to 'false' since setting it to 'true' would require
   that revocation information be available to the pledge and this
   document does not make normative requirements for [RFC6961] or
   equivalent integrations.

expires-on: This is set for nonceless vouchers. The MASA ensures
   the voucher lifetime is consistent with any revocation or pinned-
   domain-cert consistency checks the pledge might perform. See
   section Section 2.6.1. There are three times to consider: (a) a
   configured voucher lifetime in the MASA, (b) the expiry time for
   the registrar's certificate, (c) any certificate revocation
   information (CRL) lifetime. The expires-on field SHOULD be before
   the earliest of these three values. Typically (b) will be some
   significant time in the future, but (c) will typically be short
   (on the order of a week or less). The RECOMMENDED period for (a)
   is on the order of 20 minutes, so it will typically determine the
   lifespan of the resulting voucher.

Whenever a voucher is issued the MASA MUST update the audit log
appropriately. The internal state requirements to maintain the audit
log are out-of-scope. See Section 5.8.1 for a discussion of
reporting the log to a registrar.

5.6.1. Pledge voucher verification

   The pledge MUST verify the voucher signature using the manufacturer
   installed trust anchor associated with the manufacturer's MASA (this
   is likely included in the pledge's firmware).

   The pledge MUST verify the serial-number field of the signed voucher
   matches the pledge's own serial-number.

   The pledge MUST verify that the voucher nonce field is accurate and
   matches the nonce the pledge submitted to this registrar, or that the
   voucher is nonceless (see Section 6.2).

   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 that it does
   not recognize.

5.6.2. Pledge authentication of provisional TLS connection

   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.

   If a registrar's credentials cannot 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 2.6.1. Once
   the PKIX path validation is successful the TLS connection is no
   longer provisional.

The pinned-domain-cert MAY be installed as an 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 [RFC7030]
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.

5.7. Pledge BRSKI 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 may be 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 registrar 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 JSON fields in this POST MAY be added, see Section 7.3.

5.8. Registrar audit log request

After receiving the pledge status telemetry Section 5.7, the registrar SHOULD request the MASA audit log from the MASA service.

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

The registrar SHOULD HTTP POST 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 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 implementation MAY leverage internal state to associate this request with the original, and by now already validated, voucher-request so as to avoid an extra crypto validation.

A registrar MAY request logs at future times. If the registrar generates a new request then the MASA is forced to perform the additional cryptographic operations to verify the new request.

A MASA that receives a request for a device that 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.

In order to avoid enumeration of device audit logs, MASA that return URLs SHOULD take care to make the returned URL unguessable. For instance, rather than returning 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, the MASA SHOULD return a randomly generated value (a "slug" in web parlance). The value is used to find the relevant database entry.

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/voucher-cms+json The request is a "YANG-defined JSON
   document that has been signed using a CMS 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
   CMS structure.

5.8.1. MASA audit log response

A log data file is returned consisting of all log entries associated
with the the device selected by the IDevID presented in the request.
The audit log may be truncated of old or repeated values as explained
below. The returned data is in JSON format ([RFC7951]), and the
Content-Type SHOULD be "application/json". 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')>"
     "assertion":"<the value from the voucher assertion leaf>"
     "truncated":"<the number of domainID entries truncated>"
    },
    {
     "date":"<date/time of the entry>",
     "domainID":"<anotherDomainID extracted from voucher-request>",
     "nonce":"<any nonce if supplied (or the exact string 'NULL')>"
     "assertion":"<the value from the voucher assertion leaf>"
    }
  ],
  "truncation": {
     "nonced duplicates": "<total number of entries truncated>",
     "nonceless duplicates": "<total number of entries truncated>",
     "arbitrary": "<number of domainID entries removed entirely>"
     }
}
```

Distribution of a large log is less than ideal. This structure can
be optimized as follows: Nonced or Nonceless entries for the same
domainID MAY be truncated from the log leaving only the single most
recent nonced or nonceless entry for that domainID. In the case of

truncation the 'event' truncation value SHOULD contain a count of the
number of events for this domainID that were truncated. The log
SHOULD NOT be further reduced but there could exist operational
situation where maintaining the full log is not possible. In such
situations the log MAY be arbitrarily truncated for length, with the
number of removed entries indicated as 'arbitrary'.

If the truncation count exceeds 1024 then the MASA MAY use this value
without further incrementing it.

A log where duplicate entries for the same domain have been truncated
("nonced duplicates" and/or "nonceless duplicates) could still be
acceptable for informed decisions. A log that has had "arbitrary"
truncations is less acceptable but manufacturer transparency is
better than hidden truncations.

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

5.8.2. Registrar audit log verification

Each time the Manufacturer Authorized Signing Authority (MASA) issues
a voucher, it places it into the audit log for that device. The
details are described in Section 5.8. The contents of the audit log
can express a variety of trust levels, and this section explains what
kind of trust a registrar can derive from the entries.

While the audit log provides a list of vouchers that were issued by
the MASA, the vouchers are issued in response to voucher-requests,
and it is the contents of the voucher-requests which determines how
meaningful the audit log entries are.

A registrar SHOULD use the log information to make an informed
decision regarding the continued bootstrapping of the pledge. The
exact policy is out of scope of this document as it depends on the
security requirements within the registrar domain. Equipment that is
purchased pre-owned can be expected to have an extensive history.
The following dicussion is provided to help explain the value of each
log element:

date: The date field provides the registrar an opportunity to divide
   the log around known events such as the purchase date. Depending

on context known to the registrar or administrator evens before/
after certain dates can have different levels of importance. For
example for equipment that is expected to be new, and thus have no
history, it would be a surprise to find prior entries.

domainID: If the log includes an unexpected domainID then the pledge
   could have imprinted on an unexpected domain. The registrar can
   be expected to use a variety of techniques to define "unexpected"
   ranging from white lists of prior domains to anomoly detection
   (e.g. "this device was previously bound to a different domain than
   any other device deployed"). Log entries can also be compared
   against local history logs in search of discrepancies (e.g. "this
   device was re-deployed some number of times internally but the
   external audit log shows additional re-deployments our internal
   logs are unaware of").

nonce: Nonceless entries mean the logged domainID could
   theoretically trigger a reset of the pledge and then take over
   management by using the existing nonceless voucher.

assertion: The assetion leaf in the voucher and audit log indicates
   why the MASA issued the voucher. A "verified" entry means that
   the MASA issued the associated voucher as a result of positive
   verification of ownership but this can still be problematic for
   registrar's that expected only new (not pre-owned) pledges. A
   "logged" assertion informs the registrar that the prior vouchers
   were issued with minimal verification. A "proximity" assertion
   assures the registrar that the pledge was truly communicating with
   the prior domain and thus provides assurance that the prior domain
   really has deployed the pledge.

A relatively simple policy is to white list known (internal or
external) domainIDs and to require all vouchers to have a nonce and/
or require that all nonceless vouchers be from a subset (e.g. only
internal) domainIDs. A simple action is to revoke any locally issued
credentials for the pledge in question or to refuse to forward the
voucher. 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.

5.9. 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.

An ANIMA ANI pledge MUST implement the EST automation extensions described below. 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.

5.9.1. EST Distribution of CA Certificates

The pledge SHOULD 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 5.6.1 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.

5.9.2. EST CSR Attributes

Automated bootstrapping occurs without local administrative configuration of the pledge. In some deployments it is 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 registrar MUST also confirm that the resulting CSR is formatted
as indicated before forwarding the request to a CA. If the registrar
is communicating with the CA using a protocol such as full CMC, which
provides mechanisms to override the CSR attributes, then these
mechanisms MAY be used even if the client ignores CSR Attribute
guidance.

5.9.3. EST Client Certificate Request

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

5.9.4. 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. [RFC7030] 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.

5.9.5. Multiple certificates

Pledges that require multiple certificates could establish direct EST
connections to the registrar.

5.9.6. 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.ietf-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.

    This section is considered non-normative: use suggested methods MUST
    be detailed in specific profiles of BRSKI. This is the subject for
    future work.

6.1. Trust Model

    This section explains the trust relationships detailed in
    Section 2.4:

```
+--------+         +---------+     +------------+      +------------+
|        |         |         |     |            |      |Manufacturer|
| Pledge |         | Join    |     | Domain     |      |Service     |
|        |         | Proxy   |     | Registrar  |      |(Internet)  |
|        |         |         |     |            |      |            |
+--------+         +---------+     +------------+      +------------+
```
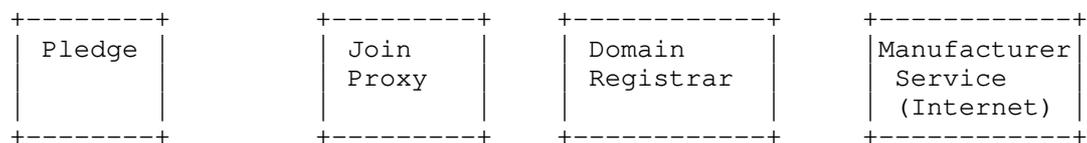
    Figure 10

    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.

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

    Registrar: When interacting with a MASA a registrar makes all
       decisions. For Ownership Audit Vouchers (see [RFC8366]) the
       registrar is provided an opportunity to accept MASA decisions.

    Vendor Service, MASA: This form of manufacturer 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 manufacturer.

    Vendor Service, Ownership Validation: This form of manufacturer
       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 a use
       case where the registrar can not connect to the MASA at the
       deployment time. Logging and validity periods address the
       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 manufacturer service is unavailable. This behavior SHOULD
       be available via local configuration or physical presence methods
       (such as use of a serial/craft console) 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 any method of skipping
   voucher validation (including use of craft serial console) 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 when voucher-based bootstrapping is not used.

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 5.5 format to ensure the pledge's serial number
information is provided to the registrar (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 the 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 the target network is protected by an air gap and
    therefore cannot 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 affect 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 a 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
    a 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 and authorized to provide this
    functionality. The MASA 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 a voucher. This is expected to be a common operational model because doing so relieves the manufacturer 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-request' information for pledges that support that functionality. This provides a proof-of-proximity check that reduces the need for ownership verification.

7. IANA Considerations

   This document requires the following IANA actions:

7.1. Well-known EST registration

   This document extends the definitions of "est" (so far defined via RFC7030) in the "https://www.iana.org/assignments/well-known-uris/well-known-uris.xhtml" registry as follows:

   o add /.well-known/est/requestvoucher (see Section 5.5)

   o add /.well-known/est/requestauditlog (see Section 5.7)

7.2. 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.

   This document requests a number from the id-pe registry (SMI Security for PKIX Certificate Extension) for id-pe-masa-url.

7.3. Pledge BRSKI Status Telemetry

   IANA is requested to create a new Registry entitled: "BRSKI Parameters", and within that Registry to create a table called: "Pledge BRSKI 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 (Section 5.7) as the
   reference:

   o version

   o Status

   o Reason

   o reason-context

7.4. DNS Service Names

   IANA is requested to register the following Service Names:

   Service Name: _brski-proxy
   Transport Protocol(s): tcp
   Assignee: IESG <iesg@ietf.org>.
   Contact: IESG <iesg@ietf.org>
   Description: The Bootstrapping Remote Secure Key
               Infrastructures Proxy
   Reference: [This document]

   Service Name: _brski-registrar
   Transport Protocol(s): tcp
   Assignee: IESG <iesg@ietf.org>.
   Contact: IESG <iesg@ietf.org>
   Description: The Bootstrapping Remote Secure Key
               Infrastructures Registrar
   Reference: [This document]

7.5. MUD File Extension for the MASA

   The IANA is requested to list the name "masa" in the MUD extensions
   registry defined in [I-D.ietf-opsawg-mud]. Its use is documented in
   Appendix C.

8. Applicability to the Autonomic Control Plane

   This document provides a solution to the requirements for secure
   bootstrap set out in Using an Autonomic Control Plane for Stable
   Connectivity of Network Operations, Administration, and Maintenance
   [RFC8368], A Reference Model for Autonomic Networking
   [I-D.ietf-anima-reference-model] and specifically the An Autonomic
   Control Plane (ACP) [I-D.ietf-anima-autonomic-control-plane], section
   3.2 (Secure Bootstrap), and section 6.1 (ACP Domain, Certificate and
   Network).

The protocol described in this document has appeal in a number of
other non-ANIMA use cases. Such uses of the protocol will be
deploying into other environments with different tradeoffs of
privacy, security, reliability and autonomy from manufacturers. As
such those use cases will need to provide their own applicability
statements, and will need to address unique privacy and security
considerations for the environments in which they are used.

The autonomic control plane that this document provides bootstrap for
is typically a medium to large Internet Service Provider
organization, or an equivalent Enterprise that has signficant layer-3
router connectivity. (A network consistenting of primarily layer-2
is not excluded, but the adjacencies that the ACP will create and
maintain will not reflect the topology until all devices participate
in the ACP).

As specified in the ANIMA charter, this work "..focuses on
professionally-managed networks." Such a network has an operator and
can do things like like install, configure and operate the Registrar
function. The operator makes purchasing decisions and is aware of
what manufacturers it expects to see on it's network.

Such an operator also is capable of performing the traditional (craft
serial-console) based bootstrap of devices. The zero-touch mechanism
presented in this and the ACP document represents a signficiant
efficiency: in particular it reduces the need to put senior experts
on airplanes to configure devices in person. There is a recognition
as the technology evolves that not every situation may work out, and
occasionally a human still still have to visit.

The BRSKI protocol is going into environments where there have
already been quite a number of vendor proprietary management systems.
Those are not expected to go away quickly, but rather to leverage the
secure credentials that are provisioned by BRSKI. The connectivity
requirements of said management systems are provided by the ACP.

9. Privacy Considerations

9.1. MASA audit log

The MASA audit 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 manufacturer 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.

9.2. Used, Stolen or Grey Market equipment

9.2.1. What BRSKI-MASA reveals to the manufacturer

The so-called "call-home" mechanism that occurs as part of the BRSKI-
MASA connection standardizes what has been deemed by some as a
sinister mechanism for corporate oversight of individuals.
([livingwithIoT] and [IoTstrangeThings] for a small sample).

As the Autonomic Control Plane (ACP) usage of BRSKI is not targetted
at individual usage of IoT devices, but rather at the Enterprise and
ISP creation of networks in a zero-touch fashion, the "call-home"
represents a different kind of concern.

First, it needs to be re-iterated that the BRSKI-MASA mechanism only
occurs once during the comissioning of the device. It is well
defined, and although encrypted with TLS, it could in theory be made
auditable as the contents are well defined.

This connection does not occur when the device powers on or is
restarted for normal routines. It is possible with low-quality
implementations that it might occur after certain firmware upgrades,
for instance if the upgrade requires repartitoning of firmware space
and this affects the credential storage. The usage of a Trusted
Platform Module (TPM) to store credentials should completely
eliminate this fault, however. Therefore this situation is clearly a
quality of implementation issue, and even in the worse case, it
results in additional enrollments being recorded when the significant
firmware update occurs.

(Some manufacturers might regard such an additional call-home a
significant feature as it might tell them how many of their customers
have performed this major upgrade, and therefore how many are still
vulnerable to some serious issue)

Second, the BRSKI call-home mechanism is mediated via the owner's
Registrar, and the information that is transmitted is directly
auditable by the device owner. This is in stark constrast to many
"call-home" protocols where the device autonomously calls home and
uses an undocumented protocol. While the contents of the signed part
of the pledge voucher request can not be changed, they are not
encrypted at the registrar. The contents of an unsigned voucher
request are, however, completely changeable by the Registrar. Both
are, to re-iterate, encrypted by TLS while in transit.

The BRSKI-MASA exchange reveals the following information to the
manufacturer:

o the identity of the device being enrolled (down to the serial-
   number!).

o an identity of the domain owner in the form of the domain trust
   anchor. However, this is not a global PKI anchored name within
   the WebPKI, so this identity could be pseudonymous. If there is
   sales channel integration, then the MASA will have authenticated
   the domain owner, either via pinned certificate, or perhaps
   another HTTP authentication method, as per Section 5.5.3.

o the time the device is activated,

o the IP address of the domain Owner's Registrar. For ISPs and
   Enterprises, the IP address provides very clear geolocation of the
   owner. No amount of IP address privacy extensions ([RFC4941]) can
   do anything about this, as a simple whois lookup likely identifies
   the ISP or Enterprise from the upper bits anyway. A passive
   attacker who observes the connection definitely may conclude that
   the given enterprise/ISP is a customer of the particular equipment
   vendor. The precise model that is being enrolled will remain
   private.

The above situation is to be distinguished from a residential/
individual person who registers a device from a manufacturer: that an
enterprise/ISP purchases routing products is hardly worth mentioning.
Deviations would, however, be notable.

The situation is not improved by the enterprise/ISP using
anonymization services such as ToR [Dingledine2004], as a TLS 1.2
connection will reveal the ClientCertificate used, clearly
identifying the enterprise/ISP involved. TLS 1.3 is better in this
regard, but an active attacker can still discover the parties
involved by performing a Man-In-The-Middle-Attack on the first
attempt (breaking/killing it with a TCP RST), and then letting
subsequent connection pass through.

A manufacturer could attempt to mix the BRSKI-MASA traffic in with
general traffic their site by hosting the MASA behind the same (set)
of load balancers that the companies normal marketing site is hosted
behind. This makes lots of sense from a straight capacity planning
point of view as the same set of services (and the same set of
Distributed Denial of Service mitigations) may be used.
Unfortunately, as the BRSKI-MASA connections include TLS
ClientCertificate exchanges, this may easily be observed in TLS 1.2,
and a traffic analysis may reveal it even in TLS 1.3. This does not
make such a plan irrelevant. There may be other organizational
reasons to keep the marketing site (which is often subject to
frequent redesigs, outsourcing, etc.) seperate from the MASA, which
may need to operate reliably for decades.

9.2.2. Manufacturers and Used or Stolen Equipment

   As explained above, the manufacturer receives information each time
   that a device which is in factory-default mode does a zero-touch
   bootstrap, and attempts to enroll into a domain owner's registrar.

   The manufacturer is therefore in a position to decline to issue a
   voucher if it detects that the new owner is not the same as the
   previous owner.

   1. This can be seen as a feature if the equipment is believed to
       have been stolen. If the legitimate owner notifies the
       manufacturer of the theft, then when the new owner brings the
       device up, if they use the zero-touch mechanism, the new
       (illegitimate) owner reveals their location and identity.

   2. In the case of Used equipment, the initial owner could inform the
       manufacturer of the sale, or the manufacturer may just permit
       resales unless told otherwise. In which case, the transfer of
       ownership simply occurs.

   3. A manufacturer could however decide not to issue a new voucher in
       response to a transfer of ownership. This is essentially the
       same as the stolen case, with the manufacturer having decided
       that the sale was not legitimate.

   4. There is a fourth case, if the manufacturer is providing
       protection against stolen devices. The manufacturer then has a
       responsability to protect the legitimate owner against fraudulent
       claims that the the equipment was stolen. Such a claim would
       cause the manufacturer to refuse to issue a new voucher. Should
       the device go through a deep factory reset (for instance,
       replacement of a damaged main board component, the device would
       not bootstrap.

   5. Finally, there is a fifth case: the manufacturer has decided to
       end-of-line the device, or the owner has not paid a yearly
       support amount, and the manufacturer refuses to issue new
       vouchers at that point. This last case is not new to the
       industry: many license systems are already deployed that have
       significantly worse effect.

   This section has outlined five situations in which a manufacturer
   could use the voucher system to enforce what are clearly license
   terms. A manufacturer that attempted to enforce license terms via
   vouchers would find it rather ineffective as the terms would only be
   enforced when the device is enrolled, and this is not (to repeat), a
   daily or even monthly occurrence.

9.2.3. Manufacturers and Grey market equipment

   Manufacturers of devices often sell different products into different
   regional markets. Which product is available in which market can be
   driven by price differentials, support issues (some markets may
   require manuals and tech-support to be done in the local language),
   government export regulation (such as whether strong crypto is
   permitted to be exported, or permitted to be used in a particular
   market). When an domain owner obtains a device from a different
   market (they can be new) and transfers it to a different location,
   this is called a Grey Market.

   A manufacturer could decide not to issue a voucher to an enterprise/
   ISP based upon their location. There are a number of ways which this
   could be determined: from the geolocation of the registrar, from
   sales channel knowledge about the customer, and what products are
   (un-)available in that market. If the device has a GPS the
   coordinates of the device could even be placed into an extension of
   the voucher.

   The above actions are not illegal, and not new. Many manufacturers
   have shipped crypto-weak (exportable) versions of firmware as the
   default on equipment for decades. The first task of an enterprise/
   ISP has always been to login to a manufacturer system, show one's
   "entitlement" (country informatin, proof that support payments have
   been made), and receive either a new updated firmware, or a license
   key that will activate the correct firmware.

   BRSKI permits the above process to automated (in an autonomic
   fashion), and therefore perhaps encourages this kind of
   differentiation by reducing the cost of doing it.

   An issue that manufacturers will need to deal with in the above
   automated process is when a device is shipped to one country with one

set of rules (or laws or entitlements), but the domain registry is in
another one. Which rules apply is something will have to be worked
out: the manufacturer could come to believe they are dealing with
Grey market equipment, when it is simply dealing with a global
enterprise.

9.2.4. Some mitigations for meddling by manufacturers

The most obvious mitigation is not to buy the product. Pick
manufacturers that are up-front about their policies, who do not
change them gratutiously.

A manufacturer could provide a mechanism to manage the trust anchors
and built-in certificates (IDevID) as an extension. This is a
substantial amount of work, and may be an area for future
standardization work.

Replacement of the voucher validation anchors (usually pointing to
the original manufacturer's MASA) with those of the new owner permits
the new owner to issue vouchers to subsequent owners. This would be
done by having the selling (old) owner to run a MASA.

In order to automatically find the new MASA, the mechanism describe
in this document is to look for the MASA URL extension in the IDevID.
A new owner could override this in their Registrar, or the
manufacturer could provide a mechanism to update or replace the
IDevID prior to sale.

Once the voucher trust anchor and the IDevID is replaced, then the
device will no longer trust the manufacturer in any way. When a new
owner performs a bootstrap, the device will point to a MASA that has
been chosen, and will validate vouchers from this new entity.

The BRSKI protocol depends upon a trust anchor on the device and an
identity on the device. Management of these these entities
facilitiates a few new operatonal modes without making any changes to
the BRSKI protocol. Those modes include: offline modes where the
domain owner operates an internal MASA for all devices, resell modes
where the first domain owner becomes the MASA for the next (resold-
to) domain owner, and services where an aggregator acquires a large
variety of devices, and then acts as a pseudonymized MASA for a
variety of devices from a variety of manufacturers.

Some manufacturers may wish to consider replacement of the IDevID as
an indication that the device's warantee is terminated. For others,
the privacy requiments of some deployments might consider this a
standard operating practice.

As discussed at the end of Section 5.8.1, new work could be done to
use a distributed consensus technology for the audit log. This would
permit the audit log to continue to be useful, even when there is a
chain of MASA due to changes of ownership.

10. Security Considerations

This document details a protocol for bootstrapping that balances
operational concerns against security concerns. As detailed in the
introduction, and touched on again in Section 6, the protocol allows
for reduced security modes. These attempt to deliver additional
control to the local administrator and owner in cases where less
security provides operational benefits. This section goes into more
detail about a variety of specific considerations.

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
registrar. 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.

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.

Pledges 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
vouchers 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 registrars verifying
log information will see multiple entries and take this into account
for their analytics purposes.

10.1. DoS against MASA

   There are uses cases where the MASA could be unavailable or
   uncooperative to the Registrar. They include active DoS attacks,
   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 in that MASA behavior
   might limit the ability to bootstrap 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
   bootstrap their devices.

   The issuance of nonceless vouchers themselves creates 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 and by Pledges only bootstrapping when in a
   factory default state. This reflects a balance between enabling MASA
   independence during future bootstrapping and the security of
   bootstrapping itself. Registrar control over requesting and auditing
   nonceless vouchers allows device owners to choose an appropriate
   balance.

   The MASA is exposed to DoS attacks wherein attackers claim an
   unbounded number of devices. Ensuring a registrar is representative
   of a valid manufacturer 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-request 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. Supply chain integration ("know your customer") is an
   additional step that MASA providers and device vendors can explore.

10.2. 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 registrar 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 registrar verifying the audit
logs available from the MASA as described in Section 5.8. Rm might
chose to collect a voucher-request but wait until after the intended
registrar completes the authorization process before submitting it.
This pledge voucher-request would be 'stale' in that it has a nonce
that no longer matches the internal state of the pledge. In order to
successfully use any resulting voucher the Rm would need to remove
the stale nonce or anticipate the pledge's future nonce state.
Reducing the possibility of this is why the pledge is mandated to
generate a strong random or pseudo-random number nonce.

Additionally, in order to successfully use the resulting voucher the
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 along with nonce information for analysis.

10.3. Trusting manufacturers

   The BRSKI extensions to EST permit a new pledge to be completely
   configured with domain specific trust anchors. The link from built-
   in manufacturer-provided trust anchors to domain-specific trust
   anchors is mediated by the signed voucher artifact.

   If the manufacturer's IDevID signing key is not properly validated,
   then there is a risk that the network will accept a pledge that
   should not be a member of the network. As the address of the
   manufacturer's MASA is provided in the IDevID using the extension
   from Section 2.3, the malicious pledge will have no problem
   collaborating with it's MASA to produce a completely valid voucher.

   BRSKI does not, however, fundamentally change the trust model from
   domain owner to manufacturer. Assuming that the pledge used its
   IDevID with RFC7030 EST and BRSKI, the domain (registrar) still needs
   to trust the manufacturer.

   Establishing this trust between domain and manufacturer is outside
   the scope of BRSKI. There are a number of mechanisms that can
   adopted including:

   o Manually configuring each manufacturer's trust anchor.

   o A Trust-On-First-Use (TOFU) mechanism. A human would be queried
      upon seeing a manufacturer's trust anchor for the first time, and
      then the trust anchor would be installed to the trusted store.
      There are risks with this; even if the key to name is validated
      using something like the WebPKI, there remains the possibility
      that the name is a look alike: e.g, c1sco.com, ..

   o scanning the trust anchor from a QR code that came with the
      packaging (this is really a manual TOFU mechanism)

   o some sales integration process where trust anchors are provided as
      part of the sales process, probably included in a digital packing
      "slip", or a sales invoice.

   o consortium membership, where all manufacturers of a particular
      device category (e.g, a light bulb, or a cable-modem) are signed
      by an certificate authority specifically for this. This is done
      by CableLabs today. It is used for authentication and
      authorization as part of TR-79: [docsisroot] and [TR069].

The existing WebPKI provides a reasonable anchor between manufacturer
name and public key. It authenticates the key. It does not provide
a reasonable authorization for the manufacturer, so it is not
directly useable on it's own.

11. Acknowledgements

We would like to thank the various reviewers for their input, in
particular William Atwood, Brian Carpenter, Toerless Eckert, Fuyu
Eleven, Eliot Lear, Sergey Kasatkin, Anoop Kumar, Markus Stenberg,
and Peter van der Stok

12. References

12.1. Normative References

[I-D.ietf-anima-autonomic-control-plane]
          Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic
          Control Plane (ACP)", draft-ietf-anima-autonomic-control-
          plane-18 (work in progress), August 2018.

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

[IDevID] IEEE Standard, "IEEE 802.1AR Secure Device Identifier",
          December 2009, <http://standards.ieee.org/findstds/
          standard/802.1AR-2009.html>.

[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>.

[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>.

   [RFC4519] Sciberras, A., Ed., "Lightweight Directory Access Protocol
              (LDAP): Schema for User Applications", RFC 4519,
              DOI 10.17487/RFC4519, June 2006,
              <https://www.rfc-editor.org/info/rfc4519>.

   [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>.

   [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS
              (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008,
              <https://www.rfc-editor.org/info/rfc5272>.

   [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>.

   [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and
              Verification of Domain-Based Application Service Identity
              within Internet Public Key Infrastructure Using X.509
              (PKIX) Certificates in the Context of Transport Layer
              Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March
              2011, <https://www.rfc-editor.org/info/rfc6125>.

   [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>.

   [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>.

   [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert,
              "A Voucher Artifact for Bootstrapping Protocols",
              RFC 8366, DOI 10.17487/RFC8366, May 2018,
              <https://www.rfc-editor.org/info/rfc8366>.

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

12.2. Informative References

   [Dingledine2004]
              Dingledine, R., Mathewson, N., and P. Syverson, "Tor: the
              second-generation onion router", 2004,
              <https://spec.torproject.org/tor-spec>.

   [docsisroot]
              CableLabs, "CableLabs Digital Certificate Issuance
              Service", February 2018,
              <https://www.cablelabs.com/resources/
              digital-certificate-issuance-service/>.

   [I-D.ietf-ace-coap-est]
              Stok, P., Kampanakis, P., Richardson, M., and S. Raza,
              "EST over secure CoAP (EST-coaps)", draft-ietf-ace-coap-
              est-07 (work in progress), January 2019.

   [I-D.ietf-anima-reference-model]
              Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L.,
              and J. Nobre, "A Reference Model for Autonomic
              Networking", draft-ietf-anima-reference-model-10 (work in
              progress), November 2018.

   [I-D.ietf-anima-stable-connectivity]
              Eckert, T. and M. Behringer, "Using Autonomic Control
              Plane for Stable Connectivity of Network OAM", draft-ietf-
              anima-stable-connectivity-10 (work in progress), February
              2018.

   [I-D.ietf-cbor-cddl]
              Birkholz, H., Vigano, C., and C. Bormann, "Concise data
              definition language (CDDL): a notational convention to
              express CBOR and JSON data structures", draft-ietf-cbor-
              cddl-06 (work in progress), November 2018.

   [I-D.ietf-netconf-zerotouch]
              Watsen, K., Abrahamsson, M., and I. Farrer, "Secure Zero
              Touch Provisioning (SZTP)", draft-ietf-netconf-
              zerotouch-29 (work in progress), January 2019.

   [I-D.ietf-opsawg-mud]
              Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage
              Description Specification", draft-ietf-opsawg-mud-25 (work
              in progress), June 2018.

   [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-02 (work in progress), January 2018.

   [imprinting]
              Wikipedia, "Wikipedia article: Imprinting", July 2015,
              <https://en.wikipedia.org/wiki/Imprinting_(psychology)>.

   [IoTstrangeThings]
              Internet, "IoT of toys stranger than fiction:
              Cybersecurity and data privacy update (accessed
              2018-12-02)", March 2017,
              <https://www.welivesecurity.com/2017/03/03/
              internet-of-things-security-privacy-iot-update/>.

   [livingwithIoT]
              Internet, "What is it actually like to live in a house
              filled with IoT devices? (accessed 2018-12-02)", February
              2018, <https://www.siliconrepublic.com/machines/
              iot-smart-devices-reality>.

   [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>.

   [RFC2663] Srisuresh, P. and M. Holdrege, "IP Network Address
              Translator (NAT) Terminology and Considerations",
              RFC 2663, DOI 10.17487/RFC2663, August 1999,
              <https://www.rfc-editor.org/info/rfc2663>.

   [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known
              Uniform Resource Identifiers (URIs)", RFC 5785,
              DOI 10.17487/RFC5785, April 2010,
              <https://www.rfc-editor.org/info/rfc5785>.

   [RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A.,
              Galperin, S., and C. Adams, "X.509 Internet Public Key
              Infrastructure Online Certificate Status Protocol - OCSP",
              RFC 6960, DOI 10.17487/RFC6960, June 2013,
              <https://www.rfc-editor.org/info/rfc6960>.

   [RFC6961] Pettersen, Y., "The Transport Layer Security (TLS)
              Multiple Certificate Status Request Extension", RFC 6961,
              DOI 10.17487/RFC6961, June 2013,
              <https://www.rfc-editor.org/info/rfc6961>.

   [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>.

   [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>.

   [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>.

   [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams",
              BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018,
              <https://www.rfc-editor.org/info/rfc8340>.

   [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>.

   [TR069]   Broadband Forum, "TR-69: CPE WAN Management Protocol",
              February 2018, <https://www.broadband-forum.org/
              standards-and-software/technical-specifications/
              tr-069-files-tools>.

Appendix A. IPv4 and non-ANI operations

   The secification of BRSKI in Section 4 intentionally only covers the
   mechanisms for an IPv6 pledge using Link-Local addresses. This
   section describes non-normative extensions that can be used in other
   environments.

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 Link-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

Pledge discovery of the proxy (Section 4.1) MAY be performed with
DNS-based Service Discovery [RFC6763] over Multicast DNS [RFC6762] to
discover the proxy at "_brski-proxy._tcp.local.".

Proxy discovery of the registrar (Section 4.3) MAY be performed with
DNS-based Service Discovery over Multicast DNS to discover registrars
by searching for the service "_brski-registrar._tcp.local.".

To prevent unaccceptable levels of network traffic, when using mDNS,
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.

Discovery of registrar MAY also be performed with DNS-based service
discovery by searching for the service "_brski-
registrar._tcp.example.com". In this case the domain "example.com"
is discovered as described in [RFC6763] section 11 (Appendix A.2
suggests the use of DHCP parameters).

If no local proxy or registrar service is located using the GRASP
mechanisms or the above mentioned DNS-based Service Discovery methods
the pledge MAY contact a well known manufacturer provided
bootstrapping server by performing a DNS lookup using a well known
URI such as "brski-registrar.manufacturer.example.com". The details
of the URI are manufacturer specific. Manufacturers that leverage
this method on the pledge are responsible for providing the registrar
service. Also see Section 2.7.

The current DNS services returned during each query are 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. 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> file "ietf-mud-extension@2018-02-14.yang"
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 2018-02-14 {
    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>
```

The MUD extensions string "masa" is defined, and MUST be included in
the extensions array of the mud container of a MUD file when this
extension is used.

Appendix D. 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.

D.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.

D.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-----

D.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-----
```

D.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
                Public-Key: (256 bit)
                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:
5b:
         9b:3a:e9:eb:8b:77:40:2e:aa:8c:87:98:da:39:03:c7:4e:
b6:
         9e:e3:62:7d:52:ad:c9:a6:ab:6b:71:77:d0:02:24:29:21:
02:
         31:00:e2:db:d7:9f:6d:32:db:76:d0:e4:de:d7:9c:63:fa:
c3:
         ed:5e:fb:5d:a2:7a:9d:80:a6:74:30:91:e7:84:eb:48:53:
4b:
         83:1b:ed:d6:5c:85:33:ed:1f:62:96:11:73:7a
```

D.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
                Public-Key: (256 bit)
                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:
0c:
         48:6b:d4:bd:61:d1:ee:e8:9c:f1:c2:5b:87:bb:d7:cb:9f:
34:
         9c:1b:3c:6e:93:67:eb:49:3f:f8:8c:ef:11:47:ad:33:32:
02:
         31:00:ab:d6:ec:6f:75:87:8a:ab:b9:9b:45:70:91:e1:90:
89:
         b3:0e:bb:7c:9e:e3:c9:76:5b:09:44:a2:af:ed:f0:05:3d:
be:
         95:68:20:cc:f0:d1:81:80:79:00:16:fb:b0:0c
```

D.2. Example process

   RFC-EDITOR: these examples will need to be replaced with CMS versions
   once IANA has assigned the eContentType in [RFC8366].

D.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
LTAyIiwibm9uY2UiOiJEc3M5OXNCcjNwTk1PUNlULxZWTd3IiwicHJveGlt
aXR5LXJlZ2lzdHJhci1jZXJ0IjoiTUlJQnJqQ0NBVE9nQXdJQkFnSUJekFFL
QmdncWhrak9QQUVFQXpCQ01SSXdFUVlLQ1pJbWlaUHlMR1FCR1FkaUFUCR1FQ1kyRXhH
VEFFYmdvSmtppYUprL0lzWkFFWkZnbHpZVlrFWknHpZVzVrWlhNaIVEFiQmdOVkJB
TU1GRlZ6Z1YzNSeWRXNW5JRVp2ZFNc1MFlXbkhlIVEFiQmdOVkJB
eE1USTBVm9YRFFNU1Ea3dOVEF4TVRJME5Wb3dRRekVTTUJBR0NnbVNKb21U
OGl4a0FSa1dBBU5oUVEXNW5JRGlpYW5pcQeUxHUJHUllKYzJGdVpHVnNi
V0Z1TVJJd0VBWURWUVFFREFsc2IyTmhiR2h2YzNRd1dUQVRCZ2NxaGtqqT1BR
SUJDZ2dxaGtqqT1BRTUJCd05QUFRMvpBN053MHhhTTS9RMnUxOTRGGelFNa3Ra
OTR3YUFJVjjBpL29WVFBnT0o4elc2TXddGNXorRHBiOC9wdWhPkpNWjBBVNkgv
d2ZBcFI2c3Z2dW1kNHJ5eW93MHdDDekFKQmdOVkhSTUVBakFBN0NcUdT
TTQ5QkfNREEya0FNR1lDMy9pVFVFK2V2WVljZ2JenZwNjR0M1FD
NnFFWVZZMmprRHgwNjJdU5pZlZdhYXJhM0YzMEFJa0tTRUNNNUURpMjll
ZmJUGJkERrM3RlY1kvckQ3Vjc3WGFKNm5ZQ21kRENSSTRUUclNGTkxneneHZ0
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:

```
    0:d=0  hl=4 l=1820 cons: SEQUENCE
    4:d=1  hl=2 l=   9 prim: OBJECT            :pkcs7-signed
```

```
    Data
      15:d=1  hl=4 l=1805 cons: cont [ 0 ]
      19:d=2  hl=4 l=1801 cons: SEQUENCE
      23:d=3  hl=2 l=   1 prim: INTEGER           :01
      26:d=3  hl=2 l=  15 cons: SET
      28:d=4  hl=2 l=  13 cons: SEQUENCE
      30:d=5  hl=2 l=   9 prim: OBJECT            :sha256
      41:d=5  hl=2 l=   0 prim: NULL
      43:d=3  hl=4 l= 782 cons: SEQUENCE
      47:d=4  hl=2 l=   9 prim: OBJECT            :pkcs7-data
      58:d=4  hl=4 l= 767 cons: cont [ 0 ]
      62:d=5  hl=4 l= 763 prim: OCTET STRING      :{"ietf-vouch
    er-request:voucher":{"assertion":"proximity","created-on":"2
    017-09-01","serial-number":"00-D0-E5-F2-00-02","nonce":"Dss9
    9sBr3pNMOACe-LYY7w","proximity-registrar-cert":"MIIBrjCCATOg
    AwIBAgIBAzAKBggqhkjOPQQDAzBOMRIwEAYKCZImiZPyLGQBGRYCY2ExGTAX
    BgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xHTAbBgNVBAMMFFVuc3RydW5nIEZv
    dW50YWluIENBMB4XDTE3MDkwNTAxMTI0NVoXDTE5MDkwNTAxMTI0NVowQzES
    MBAGCgmSJomT8ixkARkWAmNhMRkwFwYKCZImiZPyLGQBGRYJc2FuZGVsbWFu
    MRIwEAYDVQQDDAlsb2NhbGhvc3QwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNC
    AAQ1ZA7Nw0xSM/Q2u194FzQMktZ94waAIV0i/oVTPgOJ8zW6MwF5z+Dpb8/p
    uhObJMZ0U6H/wfApR6svlumd4ryyow0wCzAJBgNVHRMEAjAAMAoGCCqGSM49
    BAMDA2kAMGYCMQC3/iTQJ3evYYcgbXhbmzrp64t3QC6qjIeY2jkDx062nuNi
    fVKtyaara3F30AIkKSECMQDi29efbTLbdtDk3tecY/rD7V77XaJ6nYCmdDCR
    54TrSFNLgxvt1lyFM+0fYpYRc3o="}}
     829:d=3  hl=4 l= 566 cons: cont [ 0 ]
     833:d=4  hl=4 l= 562 cons: SEQUENCE
     837:d=5  hl=4 l= 439 cons: SEQUENCE
     841:d=6  hl=2 l=   3 cons: cont [ 0 ]
     843:d=7  hl=2 l=   1 prim: INTEGER           :02
     846:d=6  hl=2 l=   1 prim: INTEGER           :0C
     849:d=6  hl=2 l=  10 cons: SEQUENCE
     851:d=7  hl=2 l=   8 prim: OBJECT            :ecdsa-with-S
    HA256
     861:d=6  hl=2 l=  77 cons: SEQUENCE
     863:d=7  hl=2 l=  18 cons: SET
     865:d=8  hl=2 l=  16 cons: SEQUENCE
     867:d=9  hl=2 l=  10 prim: OBJECT            :domainCompon
    ent
     879:d=9  hl=2 l=   2 prim: IA5STRING         :ca
     883:d=7  hl=2 l=  25 cons: SET
     885:d=8  hl=2 l=  23 cons: SEQUENCE
     887:d=9  hl=2 l=  10 prim: OBJECT            :domainCompon
    ent
     899:d=9  hl=2 l=   9 prim: IA5STRING         :sandelman
     910:d=7  hl=2 l=  28 cons: SET
     912:d=8  hl=2 l=  26 cons: SEQUENCE
     914:d=9  hl=2 l=   3 prim: OBJECT            :commonName
```

```
  919:d=9  hl=2 l=  19 prim: UTF8STRING         :Unstrung Hig
hway CA
  940:d=6  hl=2 l=  32 cons: SEQUENCE
  942:d=7  hl=2 l=  13 prim: UTCTIME            :171012135252
Z
  957:d=7  hl=2 l=  15 prim: GENERALIZEDTIME    :299912310000
00Z
  974:d=6  hl=2 l=  75 cons: SEQUENCE
  976:d=7  hl=2 l=  18 cons: SET
  978:d=8  hl=2 l=  16 cons: SEQUENCE
  980:d=9  hl=2 l=  10 prim: OBJECT             :domainCompon
ent
  992:d=9  hl=2 l=   2 prim: IA5STRING          :ca
  996:d=7  hl=2 l=  25 cons: SET
  998:d=8  hl=2 l=  23 cons: SEQUENCE
 1000:d=9  hl=2 l=  10 prim: OBJECT             :domainCompon
ent
 1012:d=9  hl=2 l=   9 prim: IA5STRING          :sandelman
 1023:d=7  hl=2 l=  26 cons: SET
 1025:d=8  hl=2 l=  24 cons: SEQUENCE
 1027:d=9  hl=2 l=   3 prim: OBJECT             :commonName
 1032:d=9  hl=2 l=  17 prim: UTF8STRING         :00-D0-E5-F2-
00-02
 1051:d=6  hl=2 l=  89 cons: SEQUENCE
 1053:d=7  hl=2 l=  19 cons: SEQUENCE
 1055:d=8  hl=2 l=   7 prim: OBJECT             :id-ecPublicK
ey
 1064:d=8  hl=2 l=   8 prim: OBJECT             :prime256v1
 1074:d=7  hl=2 l=  66 prim: BIT STRING
 1142:d=6  hl=3 l= 135 cons: cont [ 3 ]
 1145:d=7  hl=3 l= 132 cons: SEQUENCE
 1148:d=8  hl=2 l=  29 cons: SEQUENCE
 1150:d=9  hl=2 l=   3 prim: OBJECT             :X509v3 Subje
ct Key Identifier
 1155:d=9  hl=2 l=  22 prim: OCTET STRING       [HEX DUMP]:04
141D311661B611509B3CFA13B6155F390BED76432A
 1179:d=8  hl=2 l=   9 cons: SEQUENCE
 1181:d=9  hl=2 l=   3 prim: OBJECT             :X509v3 Basic
 Constraints
 1186:d=9  hl=2 l=   2 prim: OCTET STRING       [HEX DUMP]:30
00
 1190:d=8  hl=2 l=  43 cons: SEQUENCE
 1192:d=9  hl=2 l=   3 prim: OBJECT             :X509v3 Subje
ct Alternative Name
 1197:d=9  hl=2 l=  36 prim: OCTET STRING       [HEX DUMP]:30
22A02006092B0601040182EE5201A0130C1130302D44302D45352D46322D
30302D3032
 1235:d=8  hl=2 l=  43 cons: SEQUENCE
```

```
 1237:d=9  hl=2 l=   9 prim: OBJECT            :1.3.6.1.4.1.
46930.2
 1248:d=9  hl=2 l=  30 prim: OCTET STRING      [HEX DUMP]:0C
1C68747470733A2F2F686967687761792E73616E64656C6D616E2E6361
 1280:d=5  hl=2 l=  10 cons: SEQUENCE
 1282:d=6  hl=2 l=   8 prim: OBJECT            :ecdsa-with-S
HA256
 1292:d=5  hl=2 l= 105 prim: BIT STRING
 1399:d=3  hl=4 l= 421 cons: SET
 1403:d=4  hl=4 l= 417 cons: SEQUENCE
 1407:d=5  hl=2 l=   1 prim: INTEGER           :01
 1410:d=5  hl=2 l=  82 cons: SEQUENCE
 1412:d=6  hl=2 l=  77 cons: SEQUENCE
 1414:d=7  hl=2 l=  18 cons: SET
 1416:d=8  hl=2 l=  16 cons: SEQUENCE
 1418:d=9  hl=2 l=  10 prim: OBJECT            :domainCompon
ent
 1430:d=9  hl=2 l=   2 prim: IA5STRING         :ca
 1434:d=7  hl=2 l=  25 cons: SET
 1436:d=8  hl=2 l=  23 cons: SEQUENCE
 1438:d=9  hl=2 l=  10 prim: OBJECT            :domainCompon
ent
 1450:d=9  hl=2 l=   9 prim: IA5STRING         :sandelman
 1461:d=7  hl=2 l=  28 cons: SET
 1463:d=8  hl=2 l=  26 cons: SEQUENCE
 1465:d=9  hl=2 l=   3 prim: OBJECT            :commonName
 1470:d=9  hl=2 l=  19 prim: UTF8STRING        :Unstrung Hig
hway CA
 1491:d=6  hl=2 l=   1 prim: INTEGER           :0C
 1494:d=5  hl=2 l=  13 cons: SEQUENCE
 1496:d=6  hl=2 l=   9 prim: OBJECT            :sha256
 1507:d=6  hl=2 l=   0 prim: NULL
 1509:d=5  hl=3 l= 228 cons: cont [ 0 ]
 1512:d=6  hl=2 l=  24 cons: SEQUENCE
 1514:d=7  hl=2 l=   9 prim: OBJECT            :contentType
 1525:d=7  hl=2 l=  11 cons: SET
 1527:d=8  hl=2 l=   9 prim: OBJECT            :pkcs7-data
 1538:d=6  hl=2 l=  28 cons: SEQUENCE
 1540:d=7  hl=2 l=   9 prim: OBJECT            :signingTime
 1551:d=7  hl=2 l=  15 cons: SET
 1553:d=8  hl=2 l=  13 prim: UTCTIME           :171012175430
Z
 1568:d=6  hl=2 l=  47 cons: SEQUENCE
 1570:d=7  hl=2 l=   9 prim: OBJECT            :messageDiges
t
 1581:d=7  hl=2 l=  34 cons: SET
 1583:d=8  hl=2 l=  32 prim: OCTET STRING      [HEX DUMP]:FE
7D72E29500F90A38E95021A215FD6D40B1629B99598177DC054AE0F9C8B6
```

```
   9F
    1617:d=6  hl=2 l= 121 cons: SEQUENCE
    1619:d=7  hl=2 l=   9 prim: OBJECT            :S/MIME Capab
   ilities
    1630:d=7  hl=2 l= 108 cons: SET
    1632:d=8  hl=2 l= 106 cons: SEQUENCE
    1634:d=9  hl=2 l=  11 cons: SEQUENCE
    1636:d=10 hl=2 l=   9 prim: OBJECT            :aes-256-cbc
    1647:d=9  hl=2 l=  11 cons: SEQUENCE
    1649:d=10 hl=2 l=   9 prim: OBJECT            :aes-192-cbc
    1660:d=9  hl=2 l=  11 cons: SEQUENCE
    1662:d=10 hl=2 l=   9 prim: OBJECT            :aes-128-cbc
    1673:d=9  hl=2 l=  10 cons: SEQUENCE
    1675:d=10 hl=2 l=   8 prim: OBJECT            :des-ede3-cbc
    1685:d=9  hl=2 l=  14 cons: SEQUENCE
    1687:d=10 hl=2 l=   8 prim: OBJECT            :rc2-cbc
    1697:d=10 hl=2 l=   2 prim: INTEGER           :80
    1701:d=9  hl=2 l=  13 cons: SEQUENCE
    1703:d=10 hl=2 l=   8 prim: OBJECT            :rc2-cbc
    1713:d=10 hl=2 l=   1 prim: INTEGER           :40
    1716:d=9  hl=2 l=   7 cons: SEQUENCE
    1718:d=10 hl=2 l=   5 prim: OBJECT            :des-cbc
    1725:d=9  hl=2 l=  13 cons: SEQUENCE
    1727:d=10 hl=2 l=   8 prim: OBJECT            :rc2-cbc
    1737:d=10 hl=2 l=   1 prim: INTEGER           :28
    1740:d=5  hl=2 l=  10 cons: SEQUENCE
    1742:d=6  hl=2 l=   8 prim: OBJECT            :ecdsa-with-S
   HA256
    1752:d=5  hl=2 l=  70 prim: OCTET STRING      [HEX DUMP]:30
   440220614CB435374FFB14E49BF12DEBD788FBE4BFDB3DDD4303CA3B074B
   D1C0C24AF0022008778C96F26CC9CA71FEC328A9EC9F61BF3B4E87781FFC
   8A6308FA19C0EC27CD
```

   The JSON contained in the voucher request:

   {"ietf-voucher-request:voucher":{"assertion":"proximity","cr
   eated-on":"2017-09-01","serial-number":"00-D0-E5-F2-00-02","
   nonce":"Dss99sBr3pNMOACe-LYY7w","proximity-registrar-cert":"
   MIIBrjCCATOgAwIBAgIBAzAKBggqhkjOPQQDAzBOMRIwEAYKCZImiZPyLGQB
   GRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xHTAbBgNVBAMMFFVu
   c3RydW5nIEZvdW50YWluIENBMB4XDTE3MDkwNTAxMTI0NVoXDTE5MDkwNTAx
   MTI0NVowQzESMBAGCgmSJomT8ixkARkWAmNhMRkwFwYKCZImiZPyLGQBGRYJ
   c2FuZGVsbWFuMRIwEAYDVQQDDAlsb2NhbGhvc3QwWTATBgcqhkjOPQIBBggq
   hkjOPQMBBwNCAAQ1ZA7Nw0xSM/Q2u194FzQMktZ94waAIV0i/oVTPgOJ8zW6
   MwF5z+Dpb8/puhObJMZ0U6H/wfApR6svlumd4ryyow0wCzAJBgNVHRMEAjAA
   MAoGCCqGSM49BAMDA2kAMGYCMQC3/iTQJ3evYYcgbXhbmzrp64t3QC6qjIeY
   2jkDx062nuNifVKtyaara3F30AIkKSECMQDi29efbTLbdtDk3tecY/rD7V77
   XaJ6nYCmdDCR54TrSFNLgxvt1lyFM+0fYpYRc3o="}}

D.2.2. Registrar to MASA

   As described in Section 5.5 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
   ZWQtdm91Y2hlci1yZXF1ZXN0IjoiTUlSEhRWUpLb1pJaHZjTkFRY0NvSUlI
   RGpDDQ0J3b0NBUUV4RHpBTkJnbGdoa2dCWlFNRUFnRUZBRENDQ0R0NTcUdT
   SWIzRFFFSFFhQ0NBdjhFZ2dNN2V5SnBaWFJtTFhadmRXTm9aWEl0Y21WeGRX
   VnpkRHAyYjNWamFGeVJqNWWamFFFVnlJanA3SW1GemVMdmlkR2x2YmlJNklu
   MGVTSXNJbU55WldGMGFpXURRiMjRpT2lJeU1ERTNMVEE1TFRFMVREQWVElpd
   YVddGc0xXTTFiV0pspY2lJJNklqQXdMVVF3TFVVMUxVWWlMVVE3TFRRBeUlpd2li
   bT1lWTJVaU9pSkVjM001T1hOQ2NqTndUazFQUVVVObExeVeFpXGQzSWl3aWNI
   SnZlR2x0YVhSNUxYSmxhMmx6ZEhhKaGNpMWpZV0xkRVFkRVYYcENNUMUFT
   U1hkRlFWbExMMXMBKYldsYVVIbE1SMZDjFKKWlExa3lSWGhJVkVtVGWFtZHZT
   bXRwVVVwcwwbHpXa0ZGV2tdGbmJJcFpelyV2xkNGRGbFhPSGhKVkVGaGVt
   ZE9Wa0pkCVFUxR1JssWjZek5TZVdSdWbEhTXpjVnSXk11QWkwZmFiSFZKKKUlU1
   Q1RVNTBXRVJVUlROlJHHdDNUbFJCZVUxVVVqWb1TlZUUkZVMVMUVhM2RP
   VkVGRlWUkpNRTVUVXYjNkUmVrVlRUpCUjBBbmJWmktiMjFVVT0dsNGWEwR1Nh
   MWRCYlU1b1RWRSnJkMFozV1U0RFdybHRhVnpkRWRyVnBKR2RWWkpPpKR2RW
   cEhWbkk1cVjBBaMVRWSkpkMFZCV1VSV1VWRkVSUZzYzJJZVRtdGlSSmgyWXpO
   UmpxRFFEWlFBVlJlJDDWJOeJGHdHFUMUFHSU1VVKQ1oyZHhhhR3RxTEFDFUlRVSkNkMDVF
   UVVGGUk1WcEJOMDUzTUhoVFRTOVJNblV4T1RSR2VsRuK5hM1JhaT1RSM1lVRkpW
   akwTDI5V1ZGGQm5UMUG80ZWxjMlRYZEdOWGg5yUkhCaU9DDOXdkV2hQWWtTpTldq
   QlZOa2d2ZDJZJaQmNGSTjjM1pzZFcxa05ISjVlVzkzTUhkRGVrRktRdWRPVmto
   U1RVVkJha0ZCCVFVFVGGdlIwTkRjVWRUUkRRRNFrRk5SUVV5YTBGGTlIIxbERUVkZZE
   TXk5cFZGGRktNMlYlyV1ZsalLoySllhR0p0ZW5kE05qUjBNUUZTTm5GcVNNVelN
   bXByUWkhd05pSnvkVTVwWmxaaGIbGhaWEpoTTBZZek1FFRRkphMHRUTlVU1VV
   UnBNamxsWm1KVVRHStkRVJyTTNSbxFrHxa3ZmaktFzVmpppjM1dHRktObTVaUTIx
   a1FFTlNOVGFJVY2x0R1RreG5lSFEodVTd4NNVyJNHJNR1ppaY0ZsU116TnZQU0o5
   ZmFDDQF0FqWXdnZ0l5TUlJQnnQ2QURBRZ0VDQWdFTU1Bb0dDQ3FHU000OUJBTUNN
   RTB4RWppBUUJnb0praWF0dy9Jc1BRRVpGWk0dDZ21TSm9tVDhp
   eGtBBUmtXQ1hOaGGGJtUmxiRzFoYmpFY01Cb0dBVVFkd3VFZXNNXpkSEoxYm1j
   Z1NHHbG5oSGRoZVVNCRFFUWdGdzB4TnpFd1M01USXhhNelV5TlRKYUddBOHlPVGs1
   TVRRJek1UUXdNREF3TUZzd1N6RVNNQkFHQ2dtU0pvbVQ4aXhrQVJrV0FtTmhN
   Umt3RndZS0NaSW1pWlB5TEdRQkdSdWpXMkJXRv2JXRnVNMm93R0FZRFZR
   UUREQkV3TUMxRU1ETUZOUzFHTWkwd01DMHdNakaTlJR0J5cUdTTTQ5QWdF
   R0NDcUdTTTQ5QXdFSEEwSUFCRW1ubUxxSMVRWcFNkSGE3ekF4SENDUUUI2azFz
   MHp1YlmU2FQN1FvbG1Odzhpb2dQNjJZK2RnZGRka01xTUFrQ2g1MjjoThabnlGGVCtj
   aFM5NnJES2hnandFOXVqZ11jd2dZUXdIUVlEVlIwT0JCWUVGGjB4Rm1HMkkVW
   Q2JQUG9UdGhWZk9RdnRRka01xTUFrR0ExVWRFd1FFTUFNd0t3WURWV2jBSQkNR
   d01xQWdCZ2tyQmdFRUFZTHVVZ0dnRXd3Uk1EQXRSRF0lRVdFJqSXRNREF0

TURJd0t3WUpLd1lCQkFHQzdsSUNCQjRNSEdoMGRIQnpPaTh2YUdsbmFIZGhl
UzV6WVc1a1pXeHRZVzR1WTJJZd0NnWUlLb1pJemowRUF3SURhUUF3WmdJZEFP
RW5VMzU1cWRiVlQ5N21xZ3hJYTlTOVlkSHU2Snp4d2x1SHU5ZkxuelJqR3p4
dWsyZnJTVC80ak84UlI2MHpNZ2l4QUt2VzdHOTFoNHydVp0RmNKSGhrSW16
RHJ0OG51UEpkbHNKUkttLdjdmQUZQYjZWUNETThOR0JnSGtBRnZ1d0RER0NB
YVl3Z2dHaUFnRUNBZFRFU01CQUdDZ21TSm9tVDhpeGtBUmtXQW1OaE1S
a3dGd1lLQ1pJbWlaUHlMR1I1CR1JZSmMyRnVaR1ZzYldGdU1Sd3dHZ1l2lEVlFR
RERCT1Zibk4wY25WdVp5Qklhdl2RvZDJGdU1lFTkJBZ0VNTUEwR0NXQ0dTQUZs
QXdRRU0FRVUFSUhrTUJnR0NTcUdTSWIzRFFFUkF6RUxCZ2txaGtpRzl3MEJC
d0V3SEFZSktvWklodmNOQVFrR1RFeFRWRBeE1qRXpPVGd5TTFvd0x3
WUpLb1pJaHZjTkFRa0VNU0lFSVA1ZBUGtsT09sUUlhSVYvVzFBc1dL
Ym1WbUJUjkoXdGU3VENXlMYWZNSGtHQ1NxR1NJYjNEUUVKRHpGNEdYJ
WUlaSUFXVURCQUVxTUFzR0NRUQUVxQUVVxTUFzR0NyaneuGnQlpRTUVB
UUl3Q2dZSUtvWklodmNOQXdjd0RnWUlLb1pJaHZjTkFXZjd0RnWUlLb1pJaHZjTkFF
cUdTSWIzRFFNQ0FnRkFNQWNGQlNzT0FwSUhNQTBHQ0NxR1NJYjNEUUVlFRqZ
b01Bb0dDQ3FHU000OUJBTUNSWN3UlFZ0VNZZFkSkw3RmNkSEJXUHg4cUNh
em9lOSsyMk56NFp3UkI5Z0FUMw3TU1DSVEEEanNzVWxaaekpxcDIva0NkNFdo
eFVoc2FDcFRGd1Bybk5sdzV3Q2tZ4UT09In19oIIBsjCCAa4wggEzoAMC
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
7iVpVI8CIQCD1u8Bkxipvigwv IDmWfjlYdJxcvozNjffq5j3UHg7Rg==

    file: examples/parboiled_vr_00-D0-E5-F2-00-02.pkcs

    The ASN1 decoding of the artifact:

        0:d=0 hl=4 l=3546 cons: SEQUENCE
        4:d=1 hl=2 l=   9 prim: OBJECT   :pkcs7-signed
    Data
       15:d=1 hl=4 l=3531 cons: cont [0]
       19:d=2 hl=4 l=3527 cons: SEQUENCE
       23:d=3 hl=2 l=   1 prim: INTEGER   :01

```
    26:d=3  hl=2 l=   15 cons: SET
    28:d=4  hl=2 l=   13 cons: SEQUENCE
    30:d=5  hl=2 l=    9 prim: OBJECT                 :sha256
    41:d=5  hl=2 l=    0 prim: NULL
    43:d=3  hl=4 l=2638 cons: SEQUENCE
    47:d=4  hl=2 l=    9 prim: OBJECT                 :pkcs7-data
    58:d=4  hl=4 l=2623 cons: cont [ 0 ]
    62:d=5  hl=4 l=2619 prim: OCTET STRING           :{"ietf-vouch
```

er-request:voucher":{"assertion":"proximity","created-on":"2
017-09-15T00:00:00.000Z","serial-number":"JADA123456789","no
nce":"abcd1234","prior-signed-voucher-request":"MIIHHQYJKoZI
hvcNAQcCoIIHDjCCBwoCAQExDzANBglghkgBZQMEAgEFADCCAw4GCSqGSIb3
DQEHAaCCAv8EggL7eyJpZXRmLXZvdWNoZXItcmVxdWVzdDp2b3VjaGVyIjp7
ImFzc2VydGlvbiI6InByb3hpbWl0eSIsImNyZWF0ZWQtb24iOiIyMDE3LTA5
LTAxIiwic2VyaWFsLW51bWJlciI6IjAwLUQwLUU1LUYyLTAwLTAyIiwibm9u
Y2UiOiJEc3M5OXNCcjNWTk1PUNlLUxlWWd3IiwicHJveGltaXR5LXJlZ2lz
dHJhci1jZXJ0IjoiTUlJQnNqQ0NBVE9nQXdJQkFnSUJekFLQmdncWdkncWhrak9Q
UVFFEQXpCCT01SSXdFQVlLQ1pJbWlaUHlMR1FCR1Z1Z1YyRXhIVEFYQmdvSmtp
YUprL0lzWkFFWkZnbHpVVrWld4dFlXNHhJVEFiQmdOVkJBTU1GRlZzZ1YzNS
eWRRXNW5JRVp2ZFFZFc1MFlYbHhbHVJRU5CQTI0WERURURURTNNRGt3TlRBVE1STBOVm9Y
RFRFNU1Ea3dOVEF4TVRJME5Wb3dRREVkTWBtbVNKb21UOGlxBR0NbVNKb21UOGlxBR21UOOG14a0FS
bU5oTVJrd0Z3WUtDWkltaVpQeUxHUxHUJHUllKY2ZGdlZpbHNiV0Z1TVJJd0VB
WURWUVFERERFsc2IyZ2h2h2YzNRd1dUbHJQdVZYhaGtpT1FRVRCZ2dxaGtqT1BR
T1BRTUJCZ05DQUFFRMVpQN053MHhhTTS9RMnUxOTRGlNzFaaOTR3YUFKVjjBp
L29WVFBnT4o4elc2TXddGNXorRHBiiOC9wWhPYkpNWjBBVNkgvd2ZBcFI2c3Zs
dW1kNHJ5eW93MHddekFKQmdOVkhSTUVBakFBMFNNVEUBakFBMFNNVEUBak5QQkFNREEy
a0FNR2MycFSYzNvPSJ9faCCAjYwggIyMIIBt6ADAgECAgEMMAoGCCqGSM49BAMCME0x
EjAQBgoJkiaJk/IsZAEZFgJjYTEZMBcGCgmSJomT8ixkARkWCXNhbmRlbG1h
bjEcMBoGA1UEAwwTVW5zdHJ1bmcgSGlnaHdheSBDQTAgFw0xNzEwMTIxMzUy
NTJaGA8yOTk5MTIzMTAwMDAwMFowSzESMBAGCgmSJomT8ixkARkWAmNhMRkw
FwYKCZImiZPyLGQBGRYJc2FuZGVsbWFuMRowGAYDVQQDDBEwMC1EMC1FNS1G
Mi0wMC0wMjBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABEmnmLR1TVpSdHa7
zAxHCCQ26k1s0zubWfSaP7QolmNw8iogP62s+NNKhu24h2lE98ZnyFT+chS9
6rDKhgjwE9ujgYcwgYQwHQYDVR0OBBYEFB0xFmG2EVCbPPoTthVfOQvtdkMq
MAkGA1UdEwQCMAAwKwYDVR0RBCQwIqAgBgkrBgEEAYLuUgGgEwwRMDAtRDAt
RTUtRjItMDAtMDIwKwYJKwYBBAGC7lICBB4MHGh0dHBzOi8vaGlnaHdheS5z
YW5kZWxtYW4uY2EwCgYIKoZIzj0EAwIDaQAwZgIxAOEnU355qdbVT97mqgxI
a9S9YdHu6JzxwluHu9fLnzScGzxuk2frST/4jO8RR60zMgIxAKvW7G91h4qr
uZtFcJHhkImzDrt8nuPJdlsJRKKv7fAFPb6VaCDM8NGBgHkAFvuwDDGCAaYw
ggGiAgEBMFIwTTESMBAGCgmSJomT8ixkARkWAmNhMRkwFwYKCZImiZPyLGQB
GRYJc2FuZGVsbWFuMRwwGgYDVQQDDBNVbnN0cnVuZyBIaWdod2F5IENBAgEM
MA0GCWCGSAFlAwQCAQUAoIHkMBgGCSqGSIb3DQEJAzELBgkqhkiG9w0BBwEw
HAYJKoZIhvcNAQkFMQ8XDTE3MTAxMjEzNTgyM1owLwYJKoZIhvcNAQkEMSIE
IP59cuKVAPkKOOlQIaIV/W1AsWKbmVmBd9wFSuD5yLafMHkGCSqGSIb3DQEJ
DzFsMGowCwYJYIZIAWUDBAEqMAsGCWCGSAFlAwQBFjALBglghkgBZQMEAQIw

        CgYIKoZIhvcNAwcwDgYIKoZIhvcNAwICAgCAMA0GCCqGSIb3DQMCAgFAMAcG
        BSsOAwIHMA0GCCqGSIb3DQMCAgEoMAoGCCqGSM49BAMCBEcwRQIgEMg1dJL7
        FcdtrVDx8qCazoe9+22Nz4ZwRB9gATGL7MMCIQDjssUlZzJqp2/kCd4WhxUh
        saCpTFwPrnNew5wCkYUF8Q=="}}
         2685:d=3 hl=4 l= 434 cons: cont [0]
         2689:d=4 hl=4 l= 430 cons: SEQUENCE
         2693:d=5 hl=4 l= 307 cons: SEQUENCE
         2697:d=6 hl=2 l=   3 cons: cont [0]
         2699:d=7 hl=2 l=   1 prim: INTEGER   :02
         2702:d=6 hl=2 l=   1 prim: INTEGER   :03
         2705:d=6 hl=2 l= 10 cons: SEQUENCE
         2707:d=7 hl=2 l=   8 prim: OBJECT   :ecdsa-with-S
        HA384
         2717:d=6 hl=2 l= 78 cons: SEQUENCE
         2719:d=7 hl=2 l= 18 cons: SET
         2721:d=8 hl=2 l= 16 cons: SEQUENCE
         2723:d=9 hl=2 l= 10 prim: OBJECT   :domainCompon
        ent
         2735:d=9 hl=2 l=   2 prim: IA5STRING   :ca
         2739:d=7 hl=2 l= 25 cons: SET
         2741:d=8 hl=2 l= 23 cons: SEQUENCE
         2743:d=9 hl=2 l= 10 prim: OBJECT   :domainCompon
        ent
         2755:d=9 hl=2 l=   9 prim: IA5STRING   :sandelman
         2766:d=7 hl=2 l= 29 cons: SET
         2768:d=8 hl=2 l= 27 cons: SEQUENCE
         2770:d=9 hl=2 l=   3 prim: OBJECT   :commonName
         2775:d=9 hl=2 l= 20 prim: UTF8STRING   :Unstrung Fou
        ntain CA
         2797:d=6 hl=2 l= 30 cons: SEQUENCE
         2799:d=7 hl=2 l= 13 prim: UTCTIME   :170905011245
        Z
         2814:d=7 hl=2 l= 13 prim: UTCTIME   :190905011245
        Z
         2829:d=6 hl=2 l= 67 cons: SEQUENCE
         2831:d=7 hl=2 l= 18 cons: SET
         2833:d=8 hl=2 l= 16 cons: SEQUENCE
         2835:d=9 hl=2 l= 10 prim: OBJECT   :domainCompon
        ent
         2847:d=9 hl=2 l=   2 prim: IA5STRING   :ca
         2851:d=7 hl=2 l= 25 cons: SET
         2853:d=8 hl=2 l= 23 cons: SEQUENCE
         2855:d=9 hl=2 l= 10 prim: OBJECT   :domainCompon
        ent
         2867:d=9 hl=2 l=   9 prim: IA5STRING   :sandelman
         2878:d=7 hl=2 l= 18 cons: SET
         2880:d=8 hl=2 l= 16 cons: SEQUENCE
         2882:d=9 hl=2 l=   3 prim: OBJECT   :commonName

```
 2887:d=9  hl=2 l=    9 prim: UTF8STRING         :localhost
 2898:d=6  hl=2 l=   89 cons: SEQUENCE
 2900:d=7  hl=2 l=   19 cons: SEQUENCE
 2902:d=8  hl=2 l=    7 prim: OBJECT             :id-ecPublicK
 ey
 2911:d=8  hl=2 l=    8 prim: OBJECT             :prime256v1
 2921:d=7  hl=2 l=   66 prim: BIT STRING
 2989:d=6  hl=2 l=   13 cons: cont [ 3 ]
 2991:d=7  hl=2 l=   11 cons: SEQUENCE
 2993:d=8  hl=2 l=    9 cons: SEQUENCE
 2995:d=9  hl=2 l=    3 prim: OBJECT             :X509v3 Basic
 Constraints
 3000:d=9  hl=2 l=    2 prim: OCTET STRING       [HEX DUMP]:30
 00
 3004:d=5  hl=2 l=   10 cons: SEQUENCE
 3006:d=6  hl=2 l=    8 prim: OBJECT             :ecdsa-with-S
 HA384
 3016:d=5  hl=2 l=  105 prim: BIT STRING
 3123:d=3  hl=4 l=  423 cons: SET
 3127:d=4  hl=4 l=  419 cons: SEQUENCE
 3131:d=5  hl=2 l=    1 prim: INTEGER            :01
 3134:d=5  hl=2 l=   83 cons: SEQUENCE
 3136:d=6  hl=2 l=   78 cons: SEQUENCE
 3138:d=7  hl=2 l=   18 cons: SET
 3140:d=8  hl=2 l=   16 cons: SEQUENCE
 3142:d=9  hl=2 l=   10 prim: OBJECT             :domainCompon
 ent
 3154:d=9  hl=2 l=    2 prim: IA5STRING          :ca
 3158:d=7  hl=2 l=   25 cons: SET
 3160:d=8  hl=2 l=   23 cons: SEQUENCE
 3162:d=9  hl=2 l=   10 prim: OBJECT             :domainCompon
 ent
 3174:d=9  hl=2 l=    9 prim: IA5STRING          :sandelman
 3185:d=7  hl=2 l=   29 cons: SET
 3187:d=8  hl=2 l=   27 cons: SEQUENCE
 3189:d=9  hl=2 l=    3 prim: OBJECT             :commonName
 3194:d=9  hl=2 l=   20 prim: UTF8STRING         :Unstrung Fou
 ntain CA
 3216:d=6  hl=2 l=    1 prim: INTEGER            :03
 3219:d=5  hl=2 l=   13 cons: SEQUENCE
 3221:d=6  hl=2 l=    9 prim: OBJECT             :sha256
 3232:d=6  hl=2 l=    0 prim: NULL
 3234:d=5  hl=3 l=  228 cons: cont [ 0 ]
 3237:d=6  hl=2 l=   24 cons: SEQUENCE
 3239:d=7  hl=2 l=    9 prim: OBJECT             :contentType
 3250:d=7  hl=2 l=   11 cons: SET
 3252:d=8  hl=2 l=    9 prim: OBJECT             :pkcs7-data
 3263:d=6  hl=2 l=   28 cons: SEQUENCE
```

```
 3265:d=7  hl=2 l=    9 prim: OBJECT            :signingTime
 3276:d=7  hl=2 l=   15 cons: SET
 3278:d=8  hl=2 l=   13 prim: UTCTIME           :171026013618
Z
 3293:d=6  hl=2 l=   47 cons: SEQUENCE
 3295:d=7  hl=2 l=    9 prim: OBJECT            :messageDiges
t
 3306:d=7  hl=2 l=   34 cons: SET
 3308:d=8  hl=2 l=   32 prim: OCTET STRING      [HEX DUMP]:44
0133BDCF6733E8EED13D323F2042F69A61E3103ACC65002696FC77A702A3
70
 3342:d=6  hl=2 l=  121 cons: SEQUENCE
 3344:d=7  hl=2 l=    9 prim: OBJECT            :S/MIME Capab
ilities
 3355:d=7  hl=2 l=  108 cons: SET
 3357:d=8  hl=2 l=  106 cons: SEQUENCE
 3359:d=9  hl=2 l=   11 cons: SEQUENCE
 3361:d=10 hl=2 l=    9 prim: OBJECT            :aes-256-cbc
 3372:d=9  hl=2 l=   11 cons: SEQUENCE
 3374:d=10 hl=2 l=    9 prim: OBJECT            :aes-192-cbc
 3385:d=9  hl=2 l=   11 cons: SEQUENCE
 3387:d=10 hl=2 l=    9 prim: OBJECT            :aes-128-cbc
 3398:d=9  hl=2 l=   10 cons: SEQUENCE
 3400:d=10 hl=2 l=    8 prim: OBJECT            :des-ede3-cbc
 3410:d=9  hl=2 l=   14 cons: SEQUENCE
 3412:d=10 hl=2 l=    8 prim: OBJECT            :rc2-cbc
 3422:d=10 hl=2 l=    2 prim: INTEGER           :80
 3426:d=9  hl=2 l=   13 cons: SEQUENCE
 3428:d=10 hl=2 l=    8 prim: OBJECT            :rc2-cbc
 3438:d=10 hl=2 l=    1 prim: INTEGER           :40
 3441:d=9  hl=2 l=    7 cons: SEQUENCE
 3443:d=10 hl=2 l=    5 prim: OBJECT            :des-cbc
 3450:d=9  hl=2 l=   13 cons: SEQUENCE
 3452:d=10 hl=2 l=    8 prim: OBJECT            :rc2-cbc
 3462:d=10 hl=2 l=    1 prim: INTEGER           :28
 3465:d=5  hl=2 l=   10 cons: SEQUENCE
 3467:d=6  hl=2 l=    8 prim: OBJECT            :ecdsa-with-S
HA256
 3477:d=5  hl=2 l=   71 prim: OCTET STRING      [HEX DUMP]:30
4502200DDA79B8F52530AA7B1854000FBCA9020A85BFCABE2A426DE9CDCE
EE2569548F02210083D6EF019318A9BE2830BC80E659F8E561D27172FA33
3637DFAB98F750783B46
```

D.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
LCJwaW5uZWQtZG9tYWluLWNlcnQiOiJNSUlCcmpDQ0FU2dBd0JQkF6
QUtCZ2dxaGtqT1BRUURBekJPTVJJd0VBWUtDWkltaVpQeUxHUJHU1lDWTJF
eEdUQVhCZ29Ka2lhSmsvSXNaQUVaRmdsellXNWtaWHtaW4xHDAaBgNVBAMME
QkFNTUZGVnVjVjM1ZFc1bklFWnpnZ3NWhYXkgQ0ECUUtTWJ1ZM1J5ZFc1bklF
VEF4TVJJME5Wb1hFVEU1TURrd05QXhNVEkwTlZvd1F6RVNNQkFHQ2dtU0pv
bVQ4aXhrQVJrV0FtTmhNUmt3RndZS0NaSW1pWlB5TEdRQkdSWWpjMkZ1WkdW
c2JXRnVNUkl3RUZZRFZRURREQWxzYjJJOaGJaHZjM1F3V1RBVEJnY3Foa2pP
UFFJQkJnZ3Foa2pPUFFNQkJ3TkNBQVExWkE3TncweFNNL1EydTE5NEZ6UU1r
dFo5NHdhhQUlWMGkvb1ZUUGdPSjh6VzZkY0Y1eitEcGI4L3B1aSk1aMFU2
SC93ZkFwUjZzdmx1bWWQ0cnl5b3cwd0N6QUpCZ05WSFJNRUFqQUFNQW9HQ0Nx
R1NNNDlCQU1EQTJrQU1HWUNNUUMzbGlUUUozZXNnYlhoYm16cnA2NHQz
UUM2cWpZVkkyamtzEeDA2Mm51TmlmVkt0eWFhcmEzRjMwUUlrS1NFQ01RRGky
OWVmYlRMYmR0RGszdGVjLWS9yRDdWNzdYYUo2bllDbWRE1I1NFRyU0ZOTGd4
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+Tf3DsThv7du00z
AjBjyOnmkkSKHsPR2JluA5c6wovuPEnNKP32daGGeFKGEHMkTInbrqipC881
/5K9Q+k=

file: examples/voucher_00-D0-E5-F2-00-02.pkcs

The ASN1 decoding of the artifact:

```
    0:d=0  hl=4 l=1756 cons: SEQUENCE
    4:d=1  hl=2 l=   9 prim: OBJECT                :pkcs7-signed
Data
```

```
  15:d=1  hl=4 l=1741 cons: cont [ 0 ]
  19:d=2  hl=4 l=1737 cons: SEQUENCE
  23:d=3  hl=2 l=   1 prim: INTEGER           :01
  26:d=3  hl=2 l=  15 cons: SET
  28:d=4  hl=2 l=  13 cons: SEQUENCE
  30:d=5  hl=2 l=   9 prim: OBJECT            :sha256
  41:d=5  hl=2 l=   0 prim: NULL
  43:d=3  hl=4 l= 784 cons: SEQUENCE
  47:d=4  hl=2 l=   9 prim: OBJECT            :pkcs7-data
  58:d=4  hl=4 l= 769 cons: cont [ 0 ]
  62:d=5  hl=4 l= 765 prim: OCTET STRING     :{"ietf-vouch
```
er:voucher":{"assertion":"logged","created-on":"2017-10-12T1
3:54:31.439-04:00","serial-number":"00-D0-E5-F2-00-02","nonc
e":"Dss99sBr3pNMOACe-LYY7w","pinned-domain-cert":"MIIBrjCCAT
OgAwIBAgIBAzAKBggqhkjOPQQDAzBOMRIwEAYKCZImiZPyLGQBGRYCY2ExGT
AXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xHTAbBgNVBAMMFFVuc3RydW5nIE
ZvdW50YWluIENBMB4XDTE3MDkwNTAxMTI0NVoXDTE5MDkwNTAxMTI0NVowQz
ESMBAGCgmSJomT8ixkARkWAmNhMRkwFwYKCZImiZPyLGQBGRYJc2FuZGVsbW
FuMRIwEAYDVQQDDAlsb2NhbGhvc3QwWTATBgcqhkjOPQIBBggqhkjOPQMBBw
NCAAQ1ZA7Nw0xSM/Q2u194FzQMktZ94waAIV0i/oVTPgOJ8zW6MwF5z+Dpb8
/puhObJMZ0U6H/wfApR6svlumd4ryyow0wCzAJBgNVHRMEAjAAMAoGCCqGSM
49BAMDA2kAMGYCMQC3/iTQJ3evYYcgbXhbmzrp64t3QC6qjIeY2jkDx062nu
NifVKtyaara3F30AIkKSECMQDi29efbTLbdtDk3tecY/rD7V77XaJ6nYCmdD
CR54TrSFNLgxvt1lyFM+0fYpYRc3o="}}
```
 831:d=3  hl=4 l= 467 cons: cont [ 0 ]
 835:d=4  hl=4 l= 463 cons: SEQUENCE
 839:d=5  hl=4 l= 342 cons: SEQUENCE
 843:d=6  hl=2 l=   3 cons: cont [ 0 ]
 845:d=7  hl=2 l=   1 prim: INTEGER           :02
 848:d=6  hl=2 l=   1 prim: INTEGER           :01
 851:d=6  hl=2 l=  10 cons: SEQUENCE
 853:d=7  hl=2 l=   8 prim: OBJECT            :ecdsa-with-S
HA256
 863:d=6  hl=2 l=  77 cons: SEQUENCE
 865:d=7  hl=2 l=  18 cons: SET
 867:d=8  hl=2 l=  16 cons: SEQUENCE
 869:d=9  hl=2 l=  10 prim: OBJECT            :domainCompon
ent
 881:d=9  hl=2 l=   2 prim: IA5STRING         :ca
 885:d=7  hl=2 l=  25 cons: SET
 887:d=8  hl=2 l=  23 cons: SEQUENCE
 889:d=9  hl=2 l=  10 prim: OBJECT            :domainCompon
ent
 901:d=9  hl=2 l=   9 prim: IA5STRING         :sandelman
 912:d=7  hl=2 l=  28 cons: SET
 914:d=8  hl=2 l=  26 cons: SEQUENCE
 916:d=9  hl=2 l=   3 prim: OBJECT            :commonName
 921:d=9  hl=2 l=  19 prim: UTF8STRING        :Unstrung Hig
```

```
   hway CA
     942:d=6  hl=2 l=  30 cons: SEQUENCE
     944:d=7  hl=2 l=  13 prim: UTCTIME            :170326161940
   Z
     959:d=7  hl=2 l=  13 prim: UTCTIME            :190326161940
   Z
     974:d=6  hl=2 l=  71 cons: SEQUENCE
     976:d=7  hl=2 l=  18 cons: SET
     978:d=8  hl=2 l=  16 cons: SEQUENCE
     980:d=9  hl=2 l=  10 prim: OBJECT            :domainCompon
   ent
     992:d=9  hl=2 l=   2 prim: IA5STRING          :ca
     996:d=7  hl=2 l=  25 cons: SET
     998:d=8  hl=2 l=  23 cons: SEQUENCE
    1000:d=9  hl=2 l=  10 prim: OBJECT            :domainCompon
   ent
    1012:d=9  hl=2 l=   9 prim: IA5STRING          :sandelman
    1023:d=7  hl=2 l=  22 cons: SET
    1025:d=8  hl=2 l=  20 cons: SEQUENCE
    1027:d=9  hl=2 l=   3 prim: OBJECT            :commonName
    1032:d=9  hl=2 l=  13 prim: UTF8STRING         :Unstrung MAS
   A
    1047:d=6  hl=2 l= 118 cons: SEQUENCE
    1049:d=7  hl=2 l=  16 cons: SEQUENCE
    1051:d=8  hl=2 l=   7 prim: OBJECT            :id-ecPublicK
   ey
    1060:d=8  hl=2 l=   5 prim: OBJECT            :secp384r1
    1067:d=7  hl=2 l=  98 prim: BIT STRING
    1167:d=6  hl=2 l=  16 cons: cont [ 3 ]
    1169:d=7  hl=2 l=  14 cons: SEQUENCE
    1171:d=8  hl=2 l=  12 cons: SEQUENCE
    1173:d=9  hl=2 l=   3 prim: OBJECT            :X509v3 Basic
    Constraints
    1178:d=9  hl=2 l=   1 prim: BOOLEAN            :255
    1181:d=9  hl=2 l=   2 prim: OCTET STRING       [HEX DUMP]:30
   00
    1185:d=5  hl=2 l=  10 cons: SEQUENCE
    1187:d=6  hl=2 l=   8 prim: OBJECT            :ecdsa-with-S
   HA256
    1197:d=5  hl=2 l= 103 prim: BIT STRING
    1302:d=3  hl=4 l= 454 cons: SET
    1306:d=4  hl=4 l= 450 cons: SEQUENCE
    1310:d=5  hl=2 l=   1 prim: INTEGER            :01
    1313:d=5  hl=2 l=  82 cons: SEQUENCE
    1315:d=6  hl=2 l=  77 cons: SEQUENCE
    1317:d=7  hl=2 l=  18 cons: SET
    1319:d=8  hl=2 l=  16 cons: SEQUENCE
    1321:d=9  hl=2 l=  10 prim: OBJECT            :domainCompon
```

```
  ent
   1333:d=9  hl=2 l=    2 prim: IA5STRING          :ca
   1337:d=7  hl=2 l=   25 cons: SET
   1339:d=8  hl=2 l=   23 cons: SEQUENCE
   1341:d=9  hl=2 l=   10 prim: OBJECT             :domainCompon
  ent
   1353:d=9  hl=2 l=    9 prim: IA5STRING          :sandelman
   1364:d=7  hl=2 l=   28 cons: SET
   1366:d=8  hl=2 l=   26 cons: SEQUENCE
   1368:d=9  hl=2 l=    3 prim: OBJECT             :commonName
   1373:d=9  hl=2 l=   19 prim: UTF8STRING         :Unstrung Hig
  hway CA
   1394:d=6  hl=2 l=    1 prim: INTEGER            :01
   1397:d=5  hl=2 l=   13 cons: SEQUENCE
   1399:d=6  hl=2 l=    9 prim: OBJECT             :sha256
   1410:d=6  hl=2 l=    0 prim: NULL
   1412:d=5  hl=3 l=  228 cons: cont [ 0 ]
   1415:d=6  hl=2 l=   24 cons: SEQUENCE
   1417:d=7  hl=2 l=    9 prim: OBJECT             :contentType
   1428:d=7  hl=2 l=   11 cons: SET
   1430:d=8  hl=2 l=    9 prim: OBJECT             :pkcs7-data
   1441:d=6  hl=2 l=   28 cons: SEQUENCE
   1443:d=7  hl=2 l=    9 prim: OBJECT             :signingTime
   1454:d=7  hl=2 l=   15 cons: SET
   1456:d=8  hl=2 l=   13 prim: UTCTIME            :171012175431
  Z
   1471:d=6  hl=2 l=   47 cons: SEQUENCE
   1473:d=7  hl=2 l=    9 prim: OBJECT             :messageDiges
  t
   1484:d=7  hl=2 l=   34 cons: SET
   1486:d=8  hl=2 l=   32 prim: OCTET STRING       [HEX DUMP]:41
  79C6EB6F1C216F0CA187C1D658C30E52E5250971103DAD9E372F90B11F8B
  1D
   1520:d=6  hl=2 l=  121 cons: SEQUENCE
   1522:d=7  hl=2 l=    9 prim: OBJECT             :S/MIME Capab
  ilities
   1533:d=7  hl=2 l=  108 cons: SET
   1535:d=8  hl=2 l=  106 cons: SEQUENCE
   1537:d=9  hl=2 l=   11 cons: SEQUENCE
   1539:d=10 hl=2 l=    9 prim: OBJECT             :aes-256-cbc
   1550:d=9  hl=2 l=   11 cons: SEQUENCE
   1552:d=10 hl=2 l=    9 prim: OBJECT             :aes-192-cbc
   1563:d=9  hl=2 l=   11 cons: SEQUENCE
   1565:d=10 hl=2 l=    9 prim: OBJECT             :aes-128-cbc
   1576:d=9  hl=2 l=   10 cons: SEQUENCE
   1578:d=10 hl=2 l=    8 prim: OBJECT             :des-ede3-cbc
   1588:d=9  hl=2 l=   14 cons: SEQUENCE
   1590:d=10 hl=2 l=    8 prim: OBJECT             :rc2-cbc
```

```
 1600:d=10 hl=2 l=   2 prim: INTEGER            :80
 1604:d=9  hl=2 l=  13 cons: SEQUENCE
 1606:d=10 hl=2 l=   8 prim: OBJECT             :rc2-cbc
 1616:d=10 hl=2 l=   1 prim: INTEGER            :40
 1619:d=9  hl=2 l=   7 cons: SEQUENCE
 1621:d=10 hl=2 l=   5 prim: OBJECT             :des-cbc
 1628:d=9  hl=2 l=  13 cons: SEQUENCE
 1630:d=10 hl=2 l=   8 prim: OBJECT             :rc2-cbc
 1640:d=10 hl=2 l=   1 prim: INTEGER            :28
 1643:d=5  hl=2 l=  10 cons: SEQUENCE
 1645:d=6  hl=2 l=   8 prim: OBJECT             :ecdsa-with-S
HA256
 1655:d=5  hl=2 l= 103 prim: OCTET STRING    [HEX DUMP]:30
6502310087389DFC090D8EDB6948FD6B7E5369A5D1EC8B7DB8676F935C9C
6E79EC2727CCFFD8D5DBF937F70EC4E1BFB76ED343B3023063C8E9E69244
8A1EC3D1D8996E03973AC28BEE3C49CD28FDF675A1867852861073244C89
DBAEA8A90BCF35FF92BD43E9
```

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

Email: Michael.H.Behringer@gmail.com

Steinthor Bjarnason
Arbor Networks

Email: sbjarnason@arbor.net

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

A Reference Model for Autonomic Networking
draft-ietf-anima-reference-model-10

Abstract

   This document describes a reference model for Autonomic Networking
   for managed networks. It defines the behaviour of an autonomic node,
   how the various elements in an autonomic context work together, and
   how autonomic services can use the infrastructure.

Status of This Memo

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

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

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

   This Internet-Draft will expire on May 27, 2019.

Table of Contents

1. Introduction

   The document "Autonomic Networking - Definitions and Design Goals"
   [RFC7575] explains the fundamental concepts behind Autonomic
   Networking, and defines the relevant terms in this space, as well as
   a high level reference model. [RFC7576] provides a gap analysis
   between traditional and autonomic approaches.

   This document defines this reference model with more detail, to allow
   for functional and protocol specifications to be developed in an
   architecturally consistent, non-overlapping manner.

   As discussed in [RFC7575], the goal of this work is not to focus
   exclusively on fully autonomic nodes or networks. In reality, most
   networks will run with some autonomic functions, while the rest of
   the network is traditionally managed. This reference model allows
   for this hybrid approach.

   For example, it is possible in an existing, non-autonomic network to
   enrol devices in a traditional way, to bring up a trust
   infrastructure with certificates. This trust infrastructure could
   then be used to automatically bring up an Autonomic Control Plane
   (ACP), and run traditional network operations over the secure and
   self-healing ACP. See [I-D.ietf-anima-stable-connectivity] for a
   description of this use case.

   The scope of this model is therefore limited to networks that are to
   some extent managed by skilled human operators, loosely referred to
   as "professionally managed" networks. Unmanaged networks raise
   additional security and trust issues that this model does not cover.

   This document describes a first, simple, implementable phase of an
   Autonomic Networking solution. It is expected that the experience
   from this phase will be used in defining updated and extended
   specifications over time. Some topics are considered architecturally

in this document, but are not yet reflected in the implementation
specifications. They are marked with an (*).

2. The Network View

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

   Figure 1 shows the high level view of an Autonomic Network. It
   consists of a number of autonomic nodes, which interact directly with
   each other. Those autonomic nodes provide a common set of
   capabilities across the network, called the "Autonomic Networking
   Infrastructure" (ANI). The ANI provides functions like naming,
   addressing, negotiation, synchronization, discovery and messaging.

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

```
+- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
:             :      Autonomic Function 1      :               :
: ASA 1       :      ASA 1       :      ASA 1   :        ASA 1  :
+- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
             :                   :               :
             :  +- - - - - - - - - - - - - +  :
             :  :   Autonomic Function 2   :  :
             :  : ASA 2       :      ASA 2  :  :
             :  +- - - - - - - - - - - - - +  :
             :                   :               :
+- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
:            Autonomic Networking Infrastructure          :
+- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
+--------+   :    +--------+   :    +--------+   :      +--------+
| Node 1 |--------| Node 2 |--------| Node 3 |----...-----| Node n |
+--------+   :    +--------+   :    +--------+   :      +--------+
```
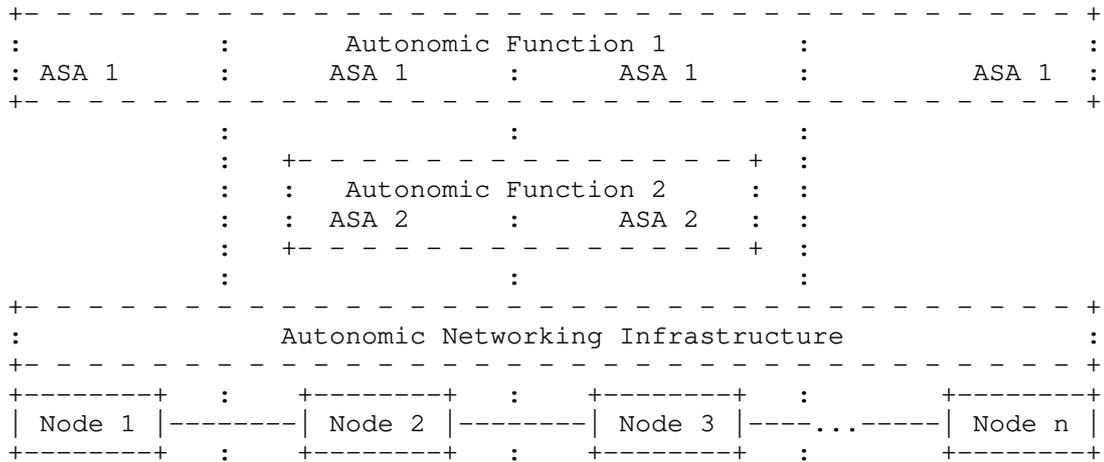
              Figure 1: High level view of an Autonomic Network

   In a horizontal view, autonomic functions span across the network, as
   well as the Autonomic Networking Infrastructure. In a vertical view,
   a node always implements the ANI, plus it may have one or several
   Autonomic Service Agents. ASAs may be standalone, or use other ASAs
   in a hierarchical way.

The Autonomic Networking Infrastructure (ANI) therefore is the
foundation for autonomic functions.

3. The Autonomic Network Element

This section explains the general architecture of an Autonomic
Network Element (Section 3.1), how it tracks its surrounding
environment in an Adjacency Table (Section 3.2), and the state
machine which defines the behaviour of the network element
(Section 3.3), based on that adjacency table.

3.1. Architecture

This section describes an autonomic network element and its internal
architecture. The reference model explained in the document
"Autonomic Networking - Definitions and Design Goals" [RFC7575] shows
the sources of information that an autonomic service agent can
leverage: Self-knowledge, network knowledge (through discovery),
Intent (see Section 7.2), and feedback loops. There are two levels
inside an autonomic node: the level of Autonomic Service Agents, and
the level of the Autonomic Networking Infrastructure, with the former
using the services of the latter. Figure 2 illustrates this concept.

```
+--------------------------------------------------------------+
|                                                              |
|  +-----------+      +-----------+        +------------+       |
|  | Autonomic |      | Autonomic |        | Autonomic  |       |
|  | Service   |      | Service   |        | Service    |       |
|  | Agent 1   |      | Agent 2   |        | Agent 3    |       |
|  +-----------+      +-----------+        +------------+       |
|        ^                 ^                    ^               |
|  - -  | -  - API level - -| - - - - - - - - - |- - -         |
|       V                  V                    V              |
|  ----------------------------------------------------------  |
| Autonomic Networking Infrastructure                          |
|     - Data structures (ex: certificates, peer information)   |
|     - Generalized Autonomic Control Plane (GACP)             |
|     - Autonomic Node Addressing and naming                   |
|     - Discovery, negotiation and synchronisation functions   |
|     - Distribution of Intent and other information           |
|     - Aggregated reporting and feedback loops                |
|     - Routing                                                |
|  ----------------------------------------------------------  |
|             Basic Operating System Functions                 |
+--------------------------------------------------------------+
```
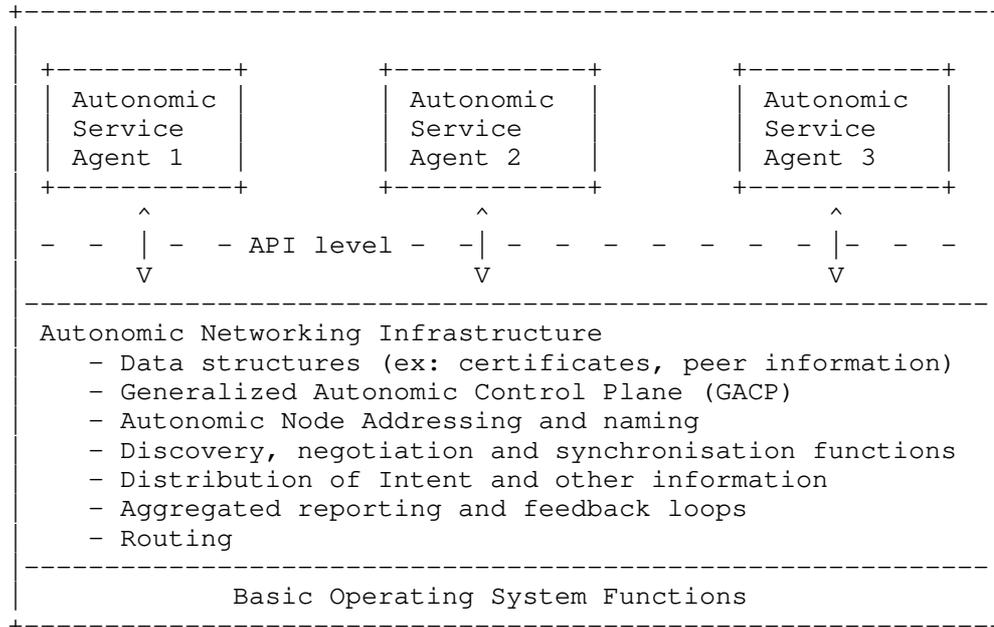
Figure 2: Model of an autonomic node

The Autonomic Networking Infrastructure (lower part of Figure 2)
contains node specific data structures, for example trust information
about itself and its peers, as well as a generic set of functions,
independent of a particular usage. This infrastructure should be
generic, and support a variety of Autonomic Service Agents (upper
part of Figure 2). It contains addressing and naming of autonomic
nodes, discovery, negotiation and synchronisation functions,
distribution of information, reporting and feedback loops, as well as
routing inside the Autonomic Control Plane.

The Generalized Autonomic Control Plane (GACP) is the summary of all
interactions of the Autonomic Networking Infrastructure with other
nodes and services. A specific implementation of the GACP is
referred to here as the Autonomic Control Plane (ACP), and described
in [I-D.ietf-anima-autonomic-control-plane].

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

The "Basic Operating System Functions" include the "normal OS",
including the network stack, security functions, etc.

Full AN nodes have the full Autonomic Networking Infrastructure, with
the full functionality described in this document. At a later stage
ANIMA may define a scope for constrained nodes with a reduced ANI and
well-defined minimal functionality. They are currently out of scope.

3.2. The Adjacency Table

Autonomic Networking is based on direct interactions between devices
of a domain. The Autonomic Control Plane (ACP) is normally
constructed on a hop-by-hop basis. Therefore, many interactions in
the ANI are based on the ANI adjacency table. There are interactions
that provide input into the adjacency table, and other interactions
that leverage the information contained in it.

The ANI adjacency table contains information about adjacent autonomic
nodes, at a minimum: node-ID, IP address in data plane, IP address in
ACP, domain, certificate. An autonomic node maintains this adjacency
table up to date. The adjacency table only contains information
about other nodes that are capable of Autonomic Networking; non-
autonomic nodes are normally not tracked here. However, the
information is tracked independently of the status of the peer nodes;
specifically, it contains information about non-enrolled nodes, nodes
of the same and other domains. The adjacency table may contain

information about the validity and trust level of the adjacent
autonomic nodes.

The adjacency table is fed by the following inputs:

o Link local discovery: This interaction happens in the data plane,
   using IPv6 link local addressing only, because this addressing
   type is itself autonomic. This way the nodes learns about all
   autonomic nodes around itself. The related standards track
   documents ([I-D.ietf-anima-grasp],
   [I-D.ietf-anima-bootstrapping-keyinfra],
   [I-D.ietf-anima-autonomic-control-plane]) describe in detail how
   link local discovery is used.

o Vendor re-direct: A new device may receive information on where
   its home network is through a vendor based Manufacturer Authorized
   Signing Authority (MASA, see Section 5.3) re-direct; this is
   typically a routable address.

o Non-autonomic input: A node may be configured manually with an
   autonomic peer; it could learn about autonomic nodes through DHCP
   options, DNS, and other non-autonomic mechanisms. Generally such
   non-autonomic mechansims require some administrator intervention.
   The key purpose is to by-pass a non-autonomic device or network.
   As this pertains to new devices, it is covered in appendix A and B
   of [I-D.ietf-anima-bootstrapping-keyinfra].

The adjacency table is defining the behaviour of an autonomic node:

o If the node has not bootstrapped into a domain (i.e., doesn't have
   a domain certificate), it rotates through all nodes in the
   adjacency table that claim to have a domain, and will attempt
   bootstrapping through them, one by one. One possible response is
   a re-direct via a vendor MASA, which will be entered into the
   adjacency table (see second bullet above). See
   [I-D.ietf-anima-bootstrapping-keyinfra] for details.

o If the adjacent node has the same domain, it will authenticate
   that adjacent node and, if successful, establish the Autonomic
   Control Plane (ACP). See
   [I-D.ietf-anima-autonomic-control-plane].

o Once the node is part of the ACP of a domain, it will use GRASP
   [I-D.ietf-anima-grasp] to find Registrar(s) of its domain and
   potentially other services.

o If the node is part of an ACP and has discovered at least one
   Registrar in its domain via GRASP, it will start the "join

     assistant" ASA, and act as a join assistant for neighboring nodes
     that need to be bootstrapped. See Section 6.3.1.2 for details.

   o Other behaviours are possible, for example establishing the ACP
      also with devices of a sub-domain, to other domains, etc. Those
      will likely be controlled by Intent. They are outside scope for
      the moment. Note that Intent is distributed through the ACP;
      therefore, a node can only adapt Intent driven behaviour once it
      has joined the ACP. At the moment, ANIMA does not consider
      providing Intent outside the ACP; this can be considered later.

   Once a node has joined the ACP, it will also learn the ACP addresses
   of its adjacent nodes, and add them to the adjacency table, to allow
   for communication inside the ACP. Further autonomic domain
   interactions will now happen inside the ACP. At this moment, only
   negotiation / synchronization via GRASP [I-D.ietf-anima-grasp] is
   being defined. (Note that GRASP runs in the data plane, as an input
   in building the adjacency table, as well as inside the ACP.)

   Autonomic Functions consist of Autonomic Service Agents (ASAs). They
   run logically above the AN Infrastructure, and may use the adjacency
   table, the ACP, negotiation and synchronization through GRASP in the
   ACP, Intent and other functions of the ANI. Since the ANI only
   provides autonomic interactions within a domain, autonomic functions
   can also use any other context on a node, specifically the global
   data plane.

3.3. State Machine

   Autonomic Networking applies during the full life-cycle of a node.
   This section describes a state machine of an autonomic node,
   throughout its life.

   A device is normally expected to store its domain specific identity,
   the LDevID (see Section 5.2), in persistent storage, to be available
   after a powercycle event. For device types that cannot store the
   LDevID in persistent storage, a powercycle event is effectively
   equivalent to a factory reset.

3.3.1. State 1: Factory Default

   An autonomic node leaves the factory in this state. In this state,
   the node has no domain specific configuration, specifically no
   LDevID, and could be used in any particular target network. It does
   however have a vendor/manufacturer specific ID, the IDevID [IDevID].
   Nodes without IDevID cannot be autonomically and securely enrolled
   into a domain; they require manual pre-staging, in which case the
   pre-staging takes them directly to state 2.

Transitions:

o Bootstrap event: The device enrols into a domain; as part of this
   process it receives a domain identity (LDevID). If enrollment is
   successful, the next state is state 2. See
   [I-D.ietf-anima-bootstrapping-keyinfra] Section 3 for details on
   enrollment.

o Powercycle event: The device loses all state tables. It remains
   in state: 1.

3.3.2. State 2: Enrolled

   An autonomic node is in the state "enrolled" if it has a domain
   identity (LDevID), and has currently no ACP channel up. It may have
   further configuration or state, for example if it had been in state 3
   before, but lost all its ACP channels. The LDevID can only be
   removed from a device through a factory reset, which also removes all
   other state from the device. This ensures that a device has no stale
   domain specific state when entering the "enrolled" state from state
   1.

   Transitions:

o Joining ACP: The device establishes an ACP channel to an adjacent
   device. See [I-D.ietf-anima-autonomic-control-plane] for details.
   Next state: 3.

o Factory reset: A factory reset removes all configuration and the
   domain identity (LDevID) from the device. Next state: 1.

o Powercycle event: The device loses all state tables, but not its
   domain identity (LDevID). it remains in state: 2.

3.3.3. State 3: In ACP

   In this state, the autonomic node has at least one ACP channel to
   another device. The node can now participate in further autonomic
   transactions, such as starting autonomic service agents (e.g., it
   must now enable the join assistant ASA, to help other devices to join
   the domain. Other conditions may apply to such interactions, for
   example to serve as a join assistant, the device must first discover
   a bootstrap Registrar.

   Transitions:

o Leaving ACP: The device drops the last (or only) ACP channel to an
   adjacent device. Next state: 2.

o Factory reset: A factory reset removes all configuration and the
   domain identity (LDevID) from the device. Next state: 1.

o Powercycle event: The device loses all state tables, but not its
   domain identity (LDevID). Next state: 2.

4. The Autonomic Networking Infrastructure

   The Autonomic Networking Infrastructure provides a layer of common
   functionality across an Autonomic Network. It provides the
   elementary functions and services, as well as extensions. An
   Autonomic Function, comprising of Autonomic Service Agents on nodes,
   uses the functions described in this section.

4.1. Naming

   Inside a domain, each autonomic device should be assigned a unique
   name. The naming scheme should be consistent within a domain. Names
   are typically assigned by a Registrar at bootstrap time and
   persistent over the lifetime of the device. All Registrars in a
   domain must follow the same naming scheme.

   In the absence of a domain specific naming scheme, a default naming
   scheme should use the same logic as the addressing scheme discussed
   in [I-D.ietf-anima-autonomic-control-plane]. The device name is then
   composed of a Registrar ID (for example taking a MAC address of the
   Registrar) and a device number. An example name would then look like
   this:

   0123-4567-89ab-0001

   The first three fields are the MAC address, the fourth field is the
   sequential number for the device.

4.2. Addressing

   Autonomic Service Agents (ASAs) need to communicate with each other,
   using the autonomic addressing of the Autonomic Networking
   Infrastructure of the node they reside on. This section describes
   the addressing approach of the Autonomic Networking Infrastructure,
   used by ASAs.

   Addressing approaches for the data plane of the network are outside
   the scope of this document. These addressing approaches may be
   configured and managed in the traditional way, or negotiated as a
   service of an ASA. One use case for such an autonomic function is
   described in [I-D.ietf-anima-prefix-management].

Autonomic addressing is a function of the Autonomic Networking
Infrastructure (lower part of Figure 2), specifically the Autonomic
Control Plane. ASAs do not have their own addresses. They may use
either API calls, or the autonomic addressing scheme of the Autonomic
Networking Infrastructure.

An autonomic addressing scheme has the following requirements:

o 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.

o 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.

o Flexibility: The addressing scheme must be flexible enough for
   nodes to be able to move around, for the network to grow, split
   and merge.

o Robustness: It should be as hard as possible for an administrator
   to negatively affect addressing (and thus connectivity) in the
   autonomic context.

o Stability: The addressing scheme should be as stable as possible.
   However, implementations need to be able to recover from
   unexpected address changes.

o Support for virtualization: Autonomic functions can exist either
   at the level of the physical network and physical devices, or at
   the level of virtual machines, containers and networks. In
   particular, Autonomic Nodes may support Autonomic Service Agents
   in virtual entities. The infrastructure, including the addressing
   scheme, should be able to support this architecture.

o 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 later phases also with autonomic nodes outside their own
domain. Therefore, an Autonomic Control Plane relies on a routing
function. For Autonomic Networks to be interoperable, they must all
support one common routing protocol.

The routing protocol is defined in the ACP document
[I-D.ietf-anima-autonomic-control-plane].

4.6. The Autonomic Control Plane

The "Autonomic Control Plane" carries the control protocols in an
autonomic network. In the architecture described here, it is
implemented as an overlay network. The document "An Autonomic
Control Plane" ([I-D.ietf-anima-autonomic-control-plane]) describes
the implementation details suggested here. This document uses the
term "overlay" to mean a set of point-to-point adjacencies congruent
with the underlying interconnection topology. The terminology may
not be aligned with a common usage of the "overlay" term in routing
context. See [I-D.ietf-anima-stable-connectivity] for uses cases for
the ACP.

4.7. Information Distribution (*)

Certain forms of information require distribution across an autonomic
domain. The distribution of information runs inside the Autonomic
Control Plane. For example, Intent is distributed across an
autonomic domain, as explained in [RFC7575].

Intent is the policy language of an Autonomic Network, see also
Section 7.2. It is a high level policy, and should change only
infrequently (order of days). Therefore, information such as Intent
should be simply flooded to all nodes in an autonomic domain, and
there is currently no perceived need to have more targeted
distribution methods. Intent is also expected to be monolithic, and
flooded as a whole. One possible method for distributing Intent, as
well as other forms of data, is discussed in
[I-D.liu-anima-grasp-distribution]. Intent and information
distribution are not part of phase 1 of ANIMA.

5. Security and Trust Infrastructure

   An Autonomic Network is self-protecting. All protocols are secure by
   default, without the requirement for the administrator to explicitly
   configure security, with the exception of setting up a PKI
   infrastructure.

   Autonomic nodes have direct interactions between themselves, which
   must be secured. Since an autonomic network does not rely on
   configuration, it is not an option to configure, for example, pre-
   shared keys. A trust infrastructure such as a PKI infrastructure
   must be in place. This section describes the principles of this
   trust infrastructure. In this first phase of autonomic networking, a
   device is either within the trust domain and fully trusted, or
   outside the trust domain and fully untrusted.

   The default method to automatically bring up a trust infrastructure
   is defined in the document "Bootstrapping Key Infrastructures"
   [I-D.ietf-anima-bootstrapping-keyinfra]. The ASAs required for this
   enrollment process are described in Section 6.3. An autonomic node
   must implement the enrollment and join assistant ASAs. The registrar
   ASA may be implemented only on a sub-set of nodes.

5.1. Public Key Infrastructure

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

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

5.2. Domain Certificate

   Each device in an autonomic domain uses a domain certificate (LDevID)
   to prove its identity. A new device uses its manufacturer provided
   certificate (IDevID) during bootstrap, to obtain a domain

certificate. [I-D.ietf-anima-bootstrapping-keyinfra] describes how a
new device receives a domain certificate, and the certificate format.

5.3. The MASA

The Manufacturer Authorized Signing Authority (MASA) is a trusted
service for bootstrapping devices. The purpose of the MASA is to
provide ownership tracking of devices in a domain. The MASA provides
audit, authorization, and ownership tokens to the registrar during
the bootstrap process to assist in the authentication of devices
attempting to join an Autonomic Domain, and to allow a joining device
to validate whether it is joining the correct domain. The details
for MASA service, security, and usage are defined in
[I-D.ietf-anima-bootstrapping-keyinfra].

5.4. Sub-Domains (*)

By default, sub-domains are treated as different domains. This
implies no trust between a domain and its sub-domains, and no trust
between sub-domains of the same domain. Specifically, no ACP is
built, and Intent is valid only for the domain it is defined for
explicitly.

In phase 2 of ANIMA, alternative trust models should be defined, for
example to allow full or limited trust between domain and sub-domain.

5.5. Cross-Domain Functionality (*)

By default, different domains do not interoperate, no ACP is built
and no trust is implied between them.

In the future, models can be established where other domains can be
trusted in full or for limited operations between the domains.

6. Autonomic Service Agents (ASA)

This section describes how autonomic services run on top of the
Autonomic Networking Infrastructure.

6.1. General Description of an ASA

An Autonomic Service Agent (ASA) is defined in [RFC7575] as "An agent
implemented on an autonomic node that implements an autonomic
function, either in part (in the case of a distributed function) or
whole." Thus it is a process that makes use of the features provided
by the ANI to achieve its own goals, usually including interaction
with other ASAs via the GRASP protocol [I-D.ietf-anima-grasp] or
otherwise. Of course it also interacts with the specific targets of

its function, using any suitable mechanism. Unless its function is
very simple, the ASA will need to handle overlapping asynchronous
operations. It may therefore be a quite complex piece of software in
its own right, forming part of the application layer above the ANI.
ASA design guidelines are available in
[I-D.carpenter-anima-asa-guidelines].

Thus we can distinguish at least three classes of ASAs:

o Simple ASAs with a small footprint that could run anywhere.

o Complex, possibly multi-threaded ASAs that have a significant
   resource requirement and will only run on selected nodes.

o A few 'infrastructure ASAs' that use basic ANI features in support
   of the ANI itself, which must run in all autonomic nodes. These
   are outlined in the following sections.

Autonomic nodes, and therefore their ASAs, know their own
capabilities and restrictions, derived from hardware, firmware or
pre-installed software: They are "self-aware".

The role of an autonomic node depends on Intent and on the
surrounding network behaviors, which may include forwarding
behaviors, aggregation properties, topology location, bandwidth,
tunnel or translation properties, etc. For example, a node may
decide to act as a backup node for a neighbor, if its capabilities
allow it to do so.

Following an initial discovery phase, the node properties and those
of its neighbors are the foundation of the behavior of a specific
node. A node and its ASAs have no pre-configuration for the
particular network in which they are installed.

Since all ASAs will interact with the ANI, they will depend on
appropriate application programming interfaces (APIs). It is
desirable that ASAs are portable between operating systems, so these
APIs need to be universal. An API for GRASP is described in
[I-D.ietf-anima-grasp-api].

ASAs will in general be designed and coded by experts in a particular
technology and use case, not by experts in the ANI and its
components. Also, they may be coded in a variety of programming
languages, in particular including languages that support object
constructs as well as traditional variables and structures. The APIs
should be designed with these factors in mind.

It must be possible to run ASAs as non-privileged (user space)
processes except for those (such as the infrastructure ASAs) that
necessarily require kernel privilege. Also, it is highly desirable
that ASAs can be dynamically loaded on a running node.

Since autonomic systems must be self-repairing, it is of great
importance that ASAs are coded using robust programming techniques.
All run-time error conditions must be caught, leading to suitable
minimally disruptive recovery actions, also considering a complete
restart of the ASA. Conditions such as discovery failures or
negotiation failures must be treated as routine, with the ASA
retrying the failed operation, preferably with an exponential back-
off in the case of persistent errors. When multiple threads are
started within an ASA, these threads must be monitored for failures
and hangups, and appropriate action taken. Attention must be given
to garbage collection, so that ASAs never run out of resources.
There is assumed to be no human operator - again, in the worst case,
every ASA must be capable of restarting itself.

ASAs will automatically benefit from the security provided by the
ANI, and specifically by the ACP and by GRASP. However, beyond that,
they are responsible for their own security, especially when
communicating with the specific targets of their function.
Therefore, the design of an ASA must include a security analysis
beyond 'use ANI security.'

6.2. ASA Life-Cycle Management

ASAs operating on a given ANI may come from different providers and
pursue different objectives. Management of ASAs and its interactions
with the ANI should follow the same operating principles, hence
comply to a generic life-cycle management model.

The ASA life-cycle provides standard processes to:

o install ASA: copy the ASA code onto the node and start it,

o deploy ASA: associate the ASA instance with a (some) managed
   network device(s) (or network function),

o control ASA execution: when and how an ASA executes its control
   loop.

The life-cyle will cover the sequential states below: Installation,
Deployment, Operation and the transitional states in-between. This
Life-Cycle will also define which interactions ASAs have with the ANI
in between the different states. The noticeable interactions are:

o Self-description of ASA instances at the end of deployment: its
   format needs to define the information required for the management
   of ASAs by ANI entities

o Control of ASA control-loop during the operation: a signaling has
   to carry formatted messages to control ASA execution (at least
   starting and stopping the control loop)

6.3. Specific ASAs for the Autonomic Network Infrastructure

   The following functions provide essential, required functionality in
   an autonomic network, and are therefore mandatory to implement on
   unconstrained autonomic nodes. They are described here as ASAs that
   include the underlying infrastructure components, but implementation
   details might vary.

   The first three together support the trust enrollment process
   described in Section 5. For details see
   [I-D.ietf-anima-bootstrapping-keyinfra].

6.3.1. The enrollment ASAs

6.3.1.1. The Pledge ASA

   This ASA includes the function of an autonomic node that bootstraps
   into the domain with the help of an join assitant ASA (see below).
   Such a node is known as a Pledge during the enrollment process. This
   ASA must be installed by default on all nodes that require an
   autonomic zero-touch bootstrap.

6.3.1.2. The Join Assistant ASA

   This ASA includes the function of an autonomic node that helps a non-
   enrolled, adjacent devices to enroll into the domain. This ASA must
   be installed on all nodes, although only one join assistant needs to
   be active on a given LAN. See also
   [I-D.ietf-anima-bootstrapping-keyinfra].

6.3.1.3. The Join Registrar ASA

   This ASA includes the join registrar function in an autonomic
   network. This ASA does not need to be installed on all nodes, but
   only on nodes that implement the Join Registrar function.

6.3.2. The ACP ASA

   This ASA includes the ACP function in an autonomic network. In
   particular it acts to discover other potential ACP nodes, and to
   support the establishment and teardown of ACP channels. This ASA
   must be installed on all nodes. For details see Section 4.6 and
   [I-D.ietf-anima-autonomic-control-plane].

6.3.3. The Information Distribution ASA (*)

   This ASA is currently out of scope in ANIMA, and provided here only
   as background information.

   This ASA includes the information distribution function in an
   autonomic network. In particular it acts to announce the
   availability of Intent and other information to all other autonomic
   nodes. This ASA does not need to be installed on all nodes, but only
   on nodes that implement the information distribution function. For
   details see Section 4.7.

   Note that information distribution can be implemented as a function
   in any ASA. See [I-D.liu-anima-grasp-distribution] for more details
   on how information is suggested to be distributed.

7. Management and Programmability

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

7.1. Managing a (Partially) Autonomic Network

   Autonomic management usually co-exists with traditional management
   methods in most networks. Thus, autonomic behavior will be defined
   for individual functions in most environments. Examples for overlap
   are:

   o Autonomic functions can use traditional methods and protocols
      (e.g., SNMP and NETCONF) to perform management tasks, inside and
      outside the ACP;

   o Autonomic functions can conflict with behavior enforced by the
      same traditional methods and protocols;

   o Traditional functions can use the ACP, for example if reachability
      on the data plane is not (yet) established.

   The autonomic Intent is defined at a high level of abstraction.
   However, since it is necessary to address individual managed

elements, autonomic management needs to communicate in lower-level
interactions (e.g., commands and requests). For example, it is
expected that the configuration of such elements be performed using
NETCONF and YANG modules as well as the monitoring be executed
through SNMP and MIBs.

Conflict can occur between autonomic default behavior, autonomic
Intent, traditional management methods. Conflict resolution is
achieved in autonomic management through prioritization [RFC7575].
The rationale is that manual and node-based management have a higher
priority over autonomic management. Thus, the autonomic default
behavior has the lowest priority, then comes the autonomic Intent
(medium priority), and, finally, the highest priority is taken by
node-specific network management methods, such as the use of command
line interfaces.

7.2. Intent (*)

Intent is not covered in the current implementation specifications.
This section discusses a topic for further research.

This section gives an overview of Intent, and how it is managed.
Intent and Policy-Based Network Management (PBNM) is already
described inside the IETF (e.g., PCIM) and in other SDOs (e.g., DMTF
and TMF ZOOM).

Intent can be described as an abstract, declarative, high-level
policy used to operate an autonomic domain, such as an enterprise
network [RFC7575]. Intent should be limited to high level guidance
only, thus it does not directly define a policy for every network
element separately.

Intent can be refined to lower level policies using different
approaches. This is expected in order to adapt the Intent to the
capabilities of managed devices. Intent may contain role or function
information, which can be translated to specific nodes [RFC7575].
One of the possible refinements of the Intent is using Event-
Condition-Action (ECA) rules.

Different parameters may be configured for Intent. These parameters
are usually provided by the human operator. Some of these parameters
can influence the behavior of specific autonomic functions as well as
the way the Intent is used to manage the autonomic domain.

Intent is discussed in more detail in [I-D.du-anima-an-intent].
Intent as well as other types of information are distributed via
GRASP, see [I-D.liu-anima-grasp-distribution].

7.3. Aggregated Reporting (*)

   Aggregated reporting is not covered in the current implementation
   specifications. This section discusses a topic for further research.

   An Autonomic Network should minimize the need for human intervention.
   In terms of how the network should behave, this is done through an
   autonomic Intent provided by the human administrator. In an
   analogous manner, the reports which describe the operational status
   of the network should aggregate the information produced in different
   network elements in order to present the effectiveness of autonomic
   Intent enforcement. Therefore, reporting in an autonomic network
   should happen on a network-wide basis [RFC7575].

   Multiple simultaneous events can occur in an autonomic network in the
   same way they can happen in a traditional network. However, when
   reporting to a human administrator, such events should be aggregated
   to avoid notifications about individual managed elements. In this
   context, algorithms may be used to determine what should be reported
   (e.g., filtering) and in which way and how different events are
   related to each other. Besides that, an event in an individual
   element can be compensated by changes in other elements to maintain a
   network-wide target which is described in the autonomic Intent.

   Reporting in an autonomic network may be at the same abstraction
   level as Intent. In this context, the aggregated view of current
   operational status of an autonomic network can be used to switch to
   different management modes. Despite the fact that autonomic
   management should minimize the need for user intervention, possibly
   there are some events that need to be addressed by human
   administrator actions.

7.4. Feedback Loops to NOC (*)

   Feedback loops are required in an autonomic network to allow the
   intervention of a human administrator or central control systems,
   while maintaining a default behaviour. Through a feedback loop an
   administrator must be prompted with a default action, and has the
   possibility to acknowledge or override the proposed default action.

   Uni-directional notifications to the NOC, that do not propose any
   default action, and do not allow an override as part of the
   transaction are considered like traditional notification services,
   such as syslog. They are expected to co-exist with autonomic
   methods, but are not covered in this draft.

7.5. Control Loops (*)

   Control loops are not covered in the current implementation
   specifications. This section discusses a topic for further research.

   Control loops are used in autonomic networking to provide a generic
   mechanism to enable the Autonomic System to adapt (on its own) to
   various factors that can change the goals that the autonomic network
   is trying to achieve, or how those goals are achieved. For example,
   as user needs, business goals, and the ANI itself changes, self-
   adaptation enables the ANI to change the services and resources it
   makes available to adapt to these changes.

   Control loops operate to continuously observe and collect data that
   enables the autonomic management system to understand changes to the
   behavior of the system being managed, and then provide actions to
   move the state of the system being managed toward a common goal.
   Self-adaptive systems move decision-making from static, pre-defined
   commands to dynamic processes computed at runtime.

   Most autonomic systems use a closed control loop with feedback. Such
   control loops should be able to be dynamically changed at runtime to
   adapt to changing user needs, business goals, and changes in the ANI.

7.6. APIs (*)

   APIs are not covered in the current implementation specifications.
   This section discusses a topic for further research.

   Most APIs are static, meaning that they are pre-defined and represent
   an invariant mechanism for operating with data. An Autonomic Network
   should be able to use dynamic APIs in addition to static APIs.

   A dynamic API is one that retrieves data using a generic mechanism,
   and then enables the client to navigate the retrieved data and
   operate on it. Such APIs typically use introspection and/or
   reflection. Introspection enables software to examine the type and
   properties of an object at runtime, while reflection enables a
   program to manipulate the attributes, methods, and/or metadata of an
   object.

   APIs must be able to express and preserve the semantics of data
   models. For example, software contracts [Meyer97] are based on the
   principle that a software-intensive system, such as an Autonomic
   Network, is a set of communicating components whose interaction is
   based on precisely-defined specifications of the mutual obligations
   that interacting components must respect. This typically includes
   specifying:

o pre-conditions that must be satisfied before the method can start
   execution

o post-conditions that must be satisfied when the method has
   finished execution

o invariant attributes that must not change during the execution of
   the method

7.7. Data Model (*)

   Data models are not covered in the current implementation
   specifications. This section discusses a topic for further research.

   The following definitions are adapted from
   [I-D.ietf-supa-generic-policy-data-model]:

   An information model is a representation of concepts of interest to
   an environment in a form that is independent of data repository, data
   definition language, query language, implementation language, and
   protocol. In contrast, a data model is a representation of concepts
   of interest to an environment in a form that is dependent on data
   repository, data definition language, query language, implementation
   language, and protocol (typically, but not necessarily, all three).

   The utility of an information model is to define objects and their
   relationships in a technology-neutral manner. This forms a
   consensual vocabulary that the ANI and ASAs can use. A data model is
   then a technology-specific mapping of all or part of the information
   model to be used by all or part of the system.

   A system may have multiple data models. Operational Support Systems,
   for example, typically have multiple types of repositories, such as
   SQL and NoSQL, to take advantage of the different properties of each.
   If multiple data models are required by an Autonomic System, then an
   information model should be used to ensure that the concepts of each
   data model can be related to each other without technological bias.

   A data model is essential for certain types of functions, such as a
   Model-Reference Adaptive Control Loop (MRACL). More generally, a
   data model can be used to define the objects, attributes, methods,
   and relationships of a software system (e.g., the ANI, an autonomic
   node, or an ASA). A data model can be used to help design an API, as
   well as any language used to interface to the Autonomic Network.

8. Coordination Between Autonomic Functions (*)

   Coordination between autonomic functions is not covered in the
   current implementation specifications. This section discusses a
   topic for further research.

8.1. The Coordination Problem (*)

   Different autonomic functions may conflict in setting certain
   parameters. For example, an energy efficiency function may want to
   shut down a redundant link, while a load balancing function would not
   want that to happen. The administrator must be able to understand
   and resolve such interactions, to steer autonomic network performance
   to a given (intended) operational point.

   Several interaction types may exist among autonomic functions, for
   example:

   o 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.

   o Dependency: An autonomic function cannot work without another one
      being present or accessible in the autonomic network.

   o 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:

   o 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.

   o At deploy-time, autonomic functions are not yet active/acting on
      the network. A "dynamic interaction map" is created for each
      instance of each autonomic functions and on a per resource basis,
      including the actions performed and their relationships. This map
      provides the basis to identify conflicts that will happen at run-
      time, categorize them and plan for the appropriate coordination
      strategies/mechanisms.

o At run-time, when conflicts happen, arbitration is driven by the
   coordination strategies. Also new dependencies can be observed
   and inferred, resulting in an update of the dynamic interaction
   map and adaptation of the coordination strategies and mechanisms.

Multiple coordination strategies and mechanisms exist and can be
devised. The set ranges from basic approaches such as random process
or token-based process, to approaches based on time separation and
hierarchical optimization, to more complex approaches such as multi-
objective optimization, and other control theory approaches and
algorithms family.

8.2. A Coordination Functional Block (*)

A common coordination functional block is a desirable component of
the ANIMA reference model. It provides a means to ensure network
properties and predictable performance or behavior such as stability,
and convergence, in the presence of several interacting autonomic
functions.

A common coordination function requires:

o A common description of autonomic functions, their attributes and
   life-cycle.

o A common representation of information and knowledge (e.g.,
   interaction maps).

o A common "control/command" interface between the coordination
   "agent" and the autonomic functions.

Guidelines, recommendations or BCPs can also be provided for aspects
pertaining to the coordination strategies and mechanisms.

9. Security Considerations

In this section we distinguish outsider and insider attacks. In an
outsider attack all network elements and protocols are securely
managed and operating, and an outside attacker can sniff packets in
transit, inject and replay packets. In an insider attack, the
attacker has access to an autonomic node or other means (e.g. remote
code execution in the node by exploiting ACP-independent
vulnerabilities in the node platform) to produce arbitrary payloads
on the protected ACP channels.

If a system has vulnerabilities in the implementation or operation
(configuration), an outside attacker can exploit such vulnerabilies
to become an insider attacker.

9.1. Protection Against Outsider Attacks

   Here, we assume that all systems involved in an autonomic network are
   secured and operated according to best current practices. These
   protection methods comprise traditional security implementation and
   operation methods (such as code security, strong randomization
   algorithms, strong passwords, etc.) as well as mechanisms specific to
   an autonomic network (such as a secured MASA service).

   Traditional security methods for both implementation and operation
   are outside scope for this document.

   AN specific protocols and methods must also follow traditional
   security methods, in that all packets that can be sniffed or injected
   by an outside attacker are:

   o protected against modification.

   o authenticated.

   o protected against replay attacks.

   o confidentiality protected (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
   unprotected AN messages outside the ACP are limited to a simple
   discovery method, defined in Section 2.5.2 of [I-D.ietf-anima-grasp]:
   The "Discovery Unsolicited Link-Local (DULL)" message, with detailed
   rules on its usage.

   If AN messages can be observed by a third party, they might reveal
   valuable information about network configuration, security
   precautions in use, individual users, and their traffic patterns. If
   encrypted, AN messages might still reveal some information via
   traffic analysis.

9.2. Risk of Insider Attacks

   An autonomic network consists of autonomic devices that form a
   distributed self-managing system. Devices within a domain have
   credentials issued from a common trust anchor and can use them to
   create mutual trust. This means that any device inside a trust
   domain can by default use all distributed functions in the entire
   autonomic domain in a malicious way.

   If an autonomic node or protocol has vulnerabilities or is not
   securely operated, an outside attacker has the following generic ways
   to take control of an autonomic network:

   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. Useful reviews were made by Joel Halpern, Radia Perlman,
   Tianran Zhou and Christian Hopps.

12. Contributors

   Significant contributions to this document have been made by John
   Strassner and Bing Liu from Huawei, and Pierre Peloso from Nokia.

13. References

13.1. Normative References

   [I-D.ietf-anima-autonomic-control-plane]
              Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic
              Control Plane (ACP)", draft-ietf-anima-autonomic-control-
              plane-18 (work in progress), August 2018.

   [I-D.ietf-anima-bootstrapping-keyinfra]
              Pritikin, M., Richardson, M., Behringer, M., Bjarnason,
              S., and K. Watsen, "Bootstrapping Remote Secure Key
              Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-
              keyinfra-17 (work in progress), November 2018.

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

   [IDevID]   IEEE Standard, , "IEEE 802.1AR Secure Device Identifier",
              December 2009, <http://standards.ieee.org/findstds/
              standard/802.1AR-2009.html>.

13.2. Informative References

   [I-D.carpenter-anima-asa-guidelines]
              Carpenter, B., Ciavaglia, L., Jiang, S., and P. Pierre,
              "Guidelines for Autonomic Service Agents", draft-
              carpenter-anima-asa-guidelines-05 (work in progress), June
              2018.

   [I-D.du-anima-an-intent]
              Du, Z., Jiang, S., Nobre, J., Ciavaglia, L., and M.
              Behringer, "ANIMA Intent Policy and Format", draft-du-
              anima-an-intent-05 (work in progress), February 2017.

   [I-D.ietf-anima-grasp-api]
             Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic
             Autonomic Signaling Protocol Application Program Interface
             (GRASP API)", draft-ietf-anima-grasp-api-02 (work in
             progress), June 2018.

   [I-D.ietf-anima-prefix-management]
             Jiang, S., Du, Z., and B. Carpenter, "Autonomic IPv6 Edge
             Prefix Management in Large-scale Networks", draft-ietf-
             anima-prefix-management-07 (work in progress), December
             2017.

   [I-D.ietf-anima-stable-connectivity]
             Eckert, T. and M. Behringer, "Using Autonomic Control
             Plane for Stable Connectivity of Network OAM", draft-ietf-
             anima-stable-connectivity-10 (work in progress), February
             2018.

   [I-D.ietf-supa-generic-policy-data-model]
             Halpern, J. and J. Strassner, "Generic Policy Data Model
             for Simplified Use of Policy Abstractions (SUPA)", draft-
             ietf-supa-generic-policy-data-model-04 (work in progress),
             June 2017.

   [I-D.liu-anima-grasp-distribution]
             Liu, B., Jiang, S., Xiao, X., Hecker, A., and Z.
             Despotovic, "Information Distribution in Autonomic
             Networking", draft-liu-anima-grasp-distribution-09 (work
             in progress), October 2018.

   [Meyer97] Meyer, B., "Object-Oriented Software Construction (2nd
             edition)", Prentice-Hall, ISBN 978-0136291558, 1997.

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

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

Authors' Addresses

    Michael H. Behringer (editor)

    Email: Michael.H.Behringer@gmail.com

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

    Email: brian.e.carpenter@gmail.com

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

    Email: tte@cs.fau.de

    Laurent Ciavaglia
    Nokia
    Villarceaux
    Nozay   91460
    FR

    Email: laurent.ciavaglia@nokia.com

    Jeferson Campos Nobre
    University of Vale do Rio dos Sinos
    Av. Unisinos, 950
    Sao Leopoldo   91501-970
    Brazil

    Email: jcnobre@unisinos.br

Network Working Group   B. Carpenter
Internet-Draft   Univ. of Auckland
Intended status: Informational   B. Liu, Ed.
Expires: May 28, 2018   Huawei Technologies
                                                           W. Wang
                                                           X. Gong
                                                     BUPT University
                                                   November 24, 2017

       Generic Autonomic Signaling Protocol Application Program Interface
                               (GRASP API)
                      draft-liu-anima-grasp-api-06

Abstract

   This document is a conceptual outline of the application programming
   interface (API) of the Generic Autonomic Signaling Protocol (GRASP).
   Such an API is needed for Autonomic Service Agents (ASA) calling the
   GRASP protocol module to exchange autonomic network messages with
   other ASAs.

carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.
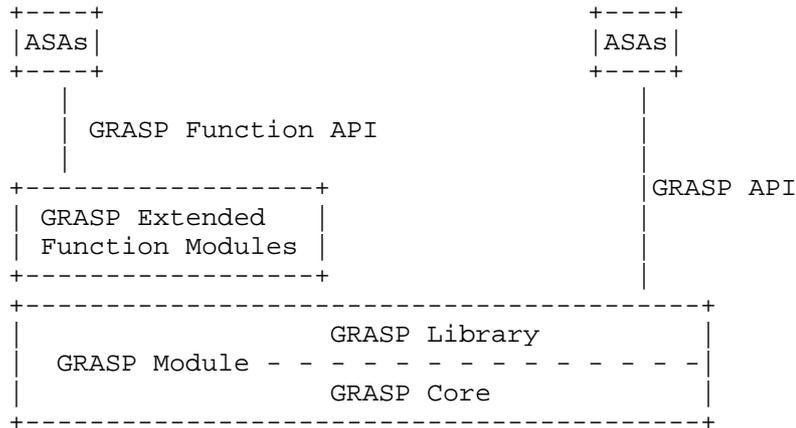
Table of Contents

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. In particular, the GRASP
library might need to communicate with the GRASP core via an inter-
process communication (IPC) mechanism.

```
          +----+                        +----+
          |ASAs|                        |ASAs|
          +----+                        +----+
            |                             |
            |  GRASP Function API         |
            |                             |
          +------------------+           |GRASP API
          | GRASP Extended   |           |
          | Function Modules |           |
          +------------------+           |
          +-------------------------------------------+
          |                  GRASP Library            |
          |   GRASP Module - - - - - - - - - - - - - -|
          |                  GRASP Core               |
          +-------------------------------------------+
```

Both the GRASP library 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.

This document gives a conceptual outline of the API. It is not a
formal specification for any particular programming language or
operating system, and it is expected that details will be clarified
in individual implementations.

2. GRASP API for ASA

2.1. Design Principles

The assumption of this document is that any Autonomic Service Agent
(ASA) needs to call a GRASP module that handles protocol details
(security, sending and listening for GRASP messages, waiting, caching

discovery results, negotiation looping, sending and receiving sychronization data, etc.) but understands nothing about individual objectives. So this is a high level abstract API for use by ASAs. Individual language bindings should be defined in separate documents.

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

o ASAs that only use GRASP for discovery purposes.

o ASAs that use GRASP negotiation but only as an initiator (client).

o ASAs that use GRASP negotiation but only as a responder.

o ASAs that use GRASP negotiation as an initiator or responder.

o ASAs that use GRASP synchronization but only as an initiator (recipient).

o ASAs that use GRASP synchronization but only as a responder and/or flooder.

o 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 mechanism for simultaneous asynchronous operations. This is discussed in detail in Section 2.2.

This is a preliminary version. A few gaps exist:

o Authorization of ASAs is out of scope.

o User-supplied explicit locators for an objective are not supported.

o The Rapid mode of GRASP is not supported.

2.2. Asynchronous Operations

GRASP includes asynchronous operations and wait states. Most ASAs will need to support several simultaneous operations; for example an ASA might need to negotiate one objective with a peer while discovering and synchronizing a different objective with a different peer. Alternatively, an ASA which acts as a resource manager might

need to run simultaneous negotiations for a given objective with
multiple different peers. Thus, both the GRASP core and most ASAs
need to support asynchronous operations. Depending on both the
operating system and the programming language in use, there are two
main techniques for such parallel operations: multi-threading, or a
polling or 'event loop' structure.

In multi-threading, the operating system and language will provide
the necessary support for asynchronous operations, including creation
of new threads, context switching between threads, queues, locks, and
implicit wait states. In this case, all API calls can be treated
naturally as synchronous, even if they include wait states, blocking
and queueing.

In an event loop implementation, synchronous blocking calls are not
acceptable. Therefore all calls must be non-blocking, and the main
loop will support multiple GRASP sessions in parallel by repeatedly
checking each one for a change of state. To facilitate this, the API
implementation will provide non-blocking versions of all the
functions that otherwise involve blocking and queueing. In these
calls, a 'noReply' code will be returned by each call instead of
blocking, until such time as the event for which it is waiting has
occurred. Thus, for example, discover() would return "noReply"
instead of waiting until discovery has succeeded or timed out. The
discover() call would be repeated in every cycle of the main loop
until it completes. A 'session_nonce' parameter (described below) is
used to distinguish simultaneous GRASP sessions from each other, so
that any number of sessions may proceed in parallel.

The following calls involve waiting for a remote operation, so they
use this mechanism:

    discover()

    request_negotiate()

    negotiate_step()

    listen_negotiate()

    synchronize()

In all these calls, the 'session_nonce' is a read/write parameter.
On the first call, it is set to a null value, and the API returns the
'noReply' code and a non-null value. This value must be used in all
subsequent calls. By this mechanism, multiple overlapping sessions
can be distinguished.

2.3. API definition

2.3.1. Parameters and data structures

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

2.3.1.1. 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 2.2).

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

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

2.3.1.2. 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.

2.3.1.3. Objective

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

   o name (UTF-8 string) - the objective's name

   o neg (Boolean flag) - True if objective supports negotiation
      (default False)

   o synch (Boolean flag) - True if objective supports synchronization
      (default False)

    o dry (Boolean flag) - True if objective 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.

    o loop_count (integer) - Limit on negotiation steps etc. (default
       GRASP_DEF_LOOPCT, see [I-D.ietf-anima-grasp])

    o 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;
     uint8_t flags;             // flag bits as defined by GRASP
     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     #True if objective supports negotiation
        self.dry = False     #True if objective supports dry-run negotiation
        self.synch = False   #True if objective supports synch
        self.loop_count = GRASP_DEF_LOOPCT  #Default starting value
        self.value = 0       #Place holder; any valid Python object
```

2.3.1.4. ASA_locator

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

   o locator - The actual locator, either an IP address or an ASCII
      string.

   o ifi (integer) - The interface identifier index via which this was
      discovered - probably no use to a normal ASA

   o expire (system dependent type) - The time on the local system
      clock when this locator will expire from the cache

   o 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

2.3.1.5. 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.

2.3.1.6. 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.

2.3.1.7. 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
   (see Section 2.2). 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.

2.3.2. 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.

   o 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.

   o 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.

o 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.

o deregister_objective()

Input parameters:

asa_nonce (integer)

objective (structure)

Return parameter:

errorcode (integer)

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

2.3.3. Discovery

o 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.

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

Event loop implementation: An additional read/write
'session_nonce' parameter is used.

2.3.4. Negotiation

o 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 negotiation.

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

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

Event loop implementation: The 'session_nonce' parameter is
used in read/write mode.

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.

o 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.

Threaded implementation: It will block waiting for an incoming
request, so should be called in a separate thread if
asynchronous operation is required.

Event loop implementation: A read/write 'session_nonce'
parameter is used.

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 negotiation.

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.

o 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'.

       Threaded implementation: Must be called from a different thread
       than 'listen_negotiate'.

       Event loop implementation: no special considerations.

   o 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.

       Threaded implementation: Called in the same thread as the
       preceding 'request_negotiate' or 'listen_negotiate', with the
       same value of 'session_nonce'.

       Event loop implementation: Must use the same value of
       'session_nonce' returned by the preceding 'request_negotiate'
       or 'listen_negotiate'.

   o negotiate_wait()

       Input parameters:

asa_nonce (integer)

session_nonce (structure)

timeout (integer)

Return parameters:

errorcode (integer)

Delay negotiation session by 'timeout' milliseconds.

Threaded implementation: Called in the same thread as the
preceding 'request_negotiate' or 'listen_negotiate', with the
same value of 'session_nonce'.

Event loop implementation: Must use the same value of
'session_nonce' returned by the preceding 'request_negotiate'
or 'listen_negotiate'.

o 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.

Threaded implementation: Called in the same thread as the
preceding 'request_negotiate' or 'listen_negotiate', with the
same value of 'session_nonce'.

Event loop implementation: Must use the same value of
'session_nonce' returned by the preceding 'request_negotiate'
or 'listen_negotiate'.

2.3.5. Synchronization and Flooding

   o 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.

         Threaded implementation: Call in a separate thread if
         asynchronous operation is required.

         Event loop implementation: An additional read/write
         'session_nonce' parameter is used.

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

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

o 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.

o 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.

o 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.

   o 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.

o 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.

2.3.6. Invalid Message Function

o 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. Example Logic Flows

   TBD

   (Until this section is written, some Python examples can be found at
   <https://github.com/becarpenter/graspy>.)

4. 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.

5. IANA Considerations

   This document does not need IANA assignments.

6. Acknowledgements

   Excellent suggestions were made by Michael Richardson and other
   participansts in the ANIMA WG.

7. References

7.1. Normative References

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

7.2. 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-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-09 (work in progress), October 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-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.

   [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-06, 2017-11-24:

    Improved description of event-loop model.

    Changed intended status to Informational.

Editorial improvements.

draft-liu-anima-grasp-api-05, 2017-10-02:

Added send_invalid()

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

Noted that simple nodes might not include the API.

Minor clarifications.

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

Changed error return to integers.

Required all implementations to accept objective values in CBOR.

Added non-blocking alternatives.

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

Updated for draft-ietf-anima-grasp-09

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

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

Editorial corrections

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

Updated for draft-ietf-anima-grasp-05

Editorial corrections

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

Initial version

Authors' Addresses

   Brian Carpenter
   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

                Information Distribution in Autonomic Networking
                    draft-liu-anima-grasp-distribution-09

Abstract

   This document discusses the requirement of capability of information
   distribution among autonomic nodes in autonomic networks. In general,
   information distribution can be categorized into two different modes:
   1) one autonomic node instantly sends information to other nodes in
   the domain; 2) one autonomic node can publish some information and
   then some other interested nodes can subscribe the published
   information. In the former case, information data will be generated
   and consumed instantly. In the latter case, however, information data
   shall be stored in the network and retrieved when necessary.

   These capabilities are fundamental and basic to a network system and
   an autonomic network infrastructure (ANI) should consider to
   integrate them, rather than assisted by other transport or routing
   protocols (HTTP, BGP/IGP as bearing protocols etc.). Thus, this
   document clarifies possible use cases and requirements to ANI so that
   information distribution can be natively supported. Possible options
   to realize the information distribution function are also briefly
   discussed.

Status of this Memo

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

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF), its areas, and its working groups. Note that
   other groups may also distribute working documents as
   Internet-Drafts.

   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."

The list of current Internet-Drafts can be accessed at
http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at
http://www.ietf.org/shadow.html

Copyright and License Notice

Table of Contents

1 Introduction

   In an autonomic network, autonomic functions (AFs) running on
   autonomic nodes would exchange information constantly, both for
   controlling/management signaling and data exchange. This document
   discusses the information distribution capability of such exchange
   between AFs.

   Scenarios of information distribution could be categorized into the
   following two basic models:

      1) Point-to-point (P2P) Communication: information are exchanged
      between two communicating parties from one node to another node.

      2) One-to-Many Communication: some information exchanges involve
      an information source and multiple receivers.

   The distribution approaches could also be categorized into two basic
   models:

      1) An instant communication: a sender connects and sends the
      information data (e.g. control/management signaling,
      synchronization data and so on) to the receiver(s) immediately.

      2) An asynchronous communication: a sender saves the information
      in the network, may or may not publish the information to the
      other who is interested in the published information, to which a
      node asks to retrieve.

   The Autonomic Network Infrastructure (ANI) [I-D.ietf-anima-reference-
   model] should have provided a generic way to support these various
   scenarios, rather than assisted by other transport or routing
   protocols (HTTP, BGP/IGP as bearing protocols etc.). In fact, GRASP
   already provides part of the capabilities.

   In this document, we first analyze requirements of information
   distribution in autonomic networks (Section 3), and then introduce

its relationship to the other modules in ANI (Section 4). After that, the node behaviors and extensions to the existing GRASP are introduced in Section 5 and Section 6, respectively.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Requirements of Advanced Information Distribution

If the information that will be exchanged is simple and short, this can be done instantly. In practice, however, this is not always the case. In the following cases, a mixture of instant and asynchronous communication models is more appropriate.

   1) Long Communication Intervals. The time interval of the communication is not necessarily always short and instant. Advanced AFs may rather involve heavy jobs/tasks (e.g. database lookup, authentication etc.) when gearing the network, so the instant mode may introduce unnecessary pending time and become less efficient. If simply using an instant mode, the AF has to wait until the tasks finish and return. A better way is that an AF instantly sends the request but switches to an synchronous mode, once the jobs are finished, AFs will get notified.

   2) Common Interest Distribution. As mentioned, some information are common interests among AFs. For example, the network intent is distributed to network nodes enrolled, which is a typical one-to-many scenario. We can also finish the intent distribution by an instant flooding (e.g. via GRASP) to every network nodes across the network domain. Because of network dynamic, however, not every node can be just ready at the moment when the network intent is flooded. Actually, nodes may join in the network sequentially. In this situation, an asynchronous communication model could be a better choice where every (newly joining) node can subscribe the intent information and will get notified if it is ready (or updated).

   3) Distributed Coordination. With computing and storage resources on autonomic nodes, alive AFs not only consumes but also generates data information. For example, AFs coordinating with each other as distributed schedulers, responding to service requests and distributing tasks. It is critical for those AFs to make correct decisions based on local information, which might be asymmetric as well. AFs may also need synthetic/aggregated data information (e.g. statistic info, like average values of several AFs, etc.) to

make decisions. In these situations, AFs will need an efficient
way to form a global view of the network (e.g. about resource
consumption, bandwidth and statistics). Obviously, purely relying
on instant communication model is inefficient, while a scalable,
common, yet distributed data layer, on which AFs can store and
share information in an asynchronous way, should be a better
choice.

For ANI, in order to support various communication scenarios, an
information distribution module is required, and both instant and
asynchronous communication models should be supported.

4. Information Distribution in ANI

This section describes how the information distribution module fits
into the ANI of ANIMA, as well as what extensions are proposed
against GRASP.

```
                         +-------------------+
                         |      ASAs         +
                         +-------------------+
                                  ^
                                  |
                                  v
 +-------------------------Info-Dist. APIs---------------------+
 |  +---------------+   +------------------+   +---------------+  |
 |  | Event Queue   |-|-| Selective Flooding|-|-| Info. Storage |  |
 |  +---------------+   +------------------+   +---------------+  |
 +--------------------------------------------------------------+
                                  ^
                                  |
                                  v
            +-------------GRASP APIs---------------+
            |  +---------------+   +---------------+  |
            |  |  GRASP Base   |-|-|   Extension   |  |
            |  +---------------+   +---------------+  |
            +--------------------------------------+
```
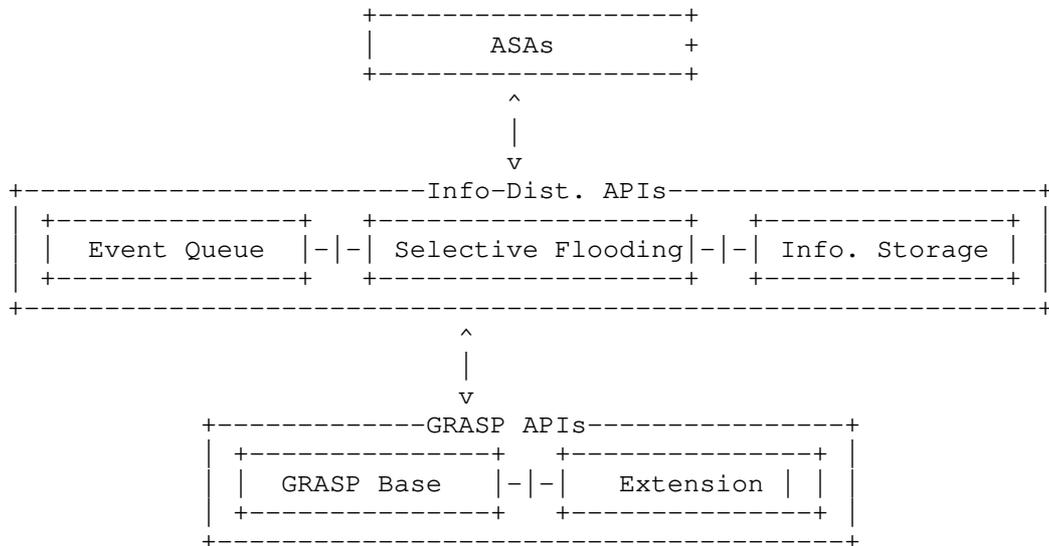
Figure 1. Information Distribution Module and GRASP Extension.

As the Fig 1 shows, the information distribution module includes
three sub-modules, all of which provides APIs for ASAs. Specific
behaviors of these modules is described in Section 5. In order to
support the modules, the GRASP is also extended, which is described
in Section 6.

5. Node Behaviors

In this section, we discuss how each autonomic node should behave in
order to realize the information distribution module. In other words,
we discuss the node requirement if an information distribution module
is required across the ANI. Supporting the two communication models
that may happen in the ANI necessarily involves node interactions and
information data exchange. Specifically, we first introduce the node
requirement for the instant communication model, and after that we
introduce the node requirement for the asynchronous communication
model.

5.1 Instant Information Distribution

In this case, sender(s) and receiver(s) are explicitly and
immediately specified (e.g. the addresses of the receivers).
Information will be directly sent from the sender(s) to the
receiver(s). This requires that every node is equipped by some
signaling/transport protocols so that they can coordinate with each
other and correctly deliver the information.

5.1.1 Instant P2P and Flooding Communications

We consider that current GRASP already provides some of the instant
P2P and flooding communications capabilities.

Straightforwardly, it is natural to use the GRASP Synchronization
message directly for P2P distribution. Furthermore, it is also
natural to use the GRASP Flood Synchronization message for 1-to-all
distribution.

However, as mentioned in Section 3, in some scenarios one node needs
to actively send some information to another. GRASP Synchronization
just lacks such capability. An un-solicited synchronization mechanism
is needed. A relevant GRASP extension is defined in Section 6.

5.1.2 Instant Selective Flooding Communication

When doing selective flooding, the distributed information needs to
contain the criteria for nodes to judge which interfaces should be
sent the distributed information and which are not. Specifically, the
criteria contain:

   o Matching condition: a set of matching rules.

   o Matching object: the object that the match condition would be
      applied to. For example, the matching object could be node itself
      or its neighbors.

   o 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.

The sender has to includes the criteria information in the message
that carries the distributed information. The receiving node decides
the action according to the criteria carried in the message. Still
considering the criteria attached with the distributed information,
the node behaviors can be:

o 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 message to the
neighbor. If not, the node discards forwarding the message to the
neighbor.

o 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 message to its
neighbors; if not, the node discards forwarding the message to the
neighbors.

An example of selective flooding is briefly described in the Appendix
A.

5.2 Asynchronous Information Distribution

Asynchronous information distribution happens in a different way
where sender(s) and receiver(s) are normally not immediately
specified. In other words, both the sender and the receiver may come
up in an asynchronous way. First of all, this requires that the
information can be stored; secondly, it requires an information
publication and subscription (Pub/Sub) mechanism. (Corresponding
protocol specification of Pub/Sub is defined in Section 6.)

As we sketched in the previous section, in general, each node
requires two modules: 1) Information Storage (IS) module and 2) Event
Queue (EQ) module. 1. These two modules should be integrated with the
information distribution module. We introduce details of the two
modules in the following sections.

5.2.1 Information Storage

IS module handles how to save and retrieve information for ASAs
across the network. The IS module uses a syntax to index information,
not only generating the hash index value (e.g. a key) for the

information, but also mapping the hash index to a certain ANIMA node
in ANI. Note that, this mechanism can use existing solutions.
Specifically, storing information in an ANIMA network will be
realized in the following steps.

1) ASA-to-IS Negotiation. An ASA calls the API provided by
information distribution module (directly supported by IS sub-
module) to request to store the information somewhere in the
network. Such a request will be checked by the IS module who will
response the request from the ASA whether such a request is
feasible according to criteria such as permitted information size.

2) Destination Node Mapping. The information block will be handled
by the IS module in order to calculate/map to a destination node
in the network. Since ANIMA network is a peer-to-peer network, a
typical way is to use dynamic hash table (DHT) to map information
to a unique index identifier. For example, if the size of the
information is reasonable, the information block itself can be
hashed, otherwise, some meta-data of the information block can be
used to generate the mapping.

3) Destination Node Negotiation Request. Negotiation request of
storing the information will be sent from the IS module to the IS
module on the destination node. The negotiation request contains
parameters about the information block from the source IS module.
According to the parameters as well as the local available
resource, the destination node will feedback the source IS module.

4) Destination Node Negotiation Response. Negotiation response
from the destination node is sent back to the source IS module. If
the source IS module gets confirmation that the information can be
stored, source IS module will prepare to transfer the information
block; otherwise, a new destination node must be discovered (i.e.
going to step 7).

5) Information Block Transfer. Before sending the information
block to the destination node that accepts the request, the IS
module of the source node will check if the information block can
be afforded by one GRASP message. If so, the information block
will be directly sent by calling a GRASP API. Otherwise, bulk data
transmission with GRASP will be triggered, where multi-time GRASP
message sending will be used so that one information block will be
transferred by smaller pieces [I-D.ietf-anima-reference-model].

6) Information Writing. Once the information block (or a smaller
block) is received, the IS module of the destination node will
store the data block in the local storage, which is accessible.

7) (Optional) New Destination Node Discovery. If the previously selected destination node is not available to store the information block, the source IS module will have to identify a new destination node to start a new negotiation. In this case, the discovery can be done by using discovery GRASP API to identify a new candidate, or more complex mechanisms can be introduced.

Similarly, Getting information from an ANIMA network will be realized in the following steps.

1) ASA-to-IS Request. An ASA accesses the IS module via the APIs exposed by the information distribution module. The key/index of the interested information will be sent to the IS module. An assumption here is that the key/index should be ready to an ASA before an ASA can ask for the information. This relates to the publishing/subscribing of the information, which are handled by other modules (e.g. Event Queue with Pub/Sub supported by GRASP).

2) Destination Node Mapping. IS module maps the key/index of the requested information to a destination node, and prepares to start to request the information. The mapping here follows the same mechanism when the information is stored.

3) Retrieval Negotiation Request. The source IS module sends a request to the destination node identified in the previous step and asks if such an information object is available.

4) Retrieval Negotiation Response. The destination node checks the key/index of the requested information, and replies to the source IS module. If the information is found and the information block can be afforded within one GRASP message, the information will be sent together with the response to the source IS module.

5) (Optional) New Destination Request. If the information is not found after the source IS module gets the response from the original destination node, the source IS module will have to discover where the location storing the requested information is.

IS module can reuse distributed databases and key value stores like NoSQL, Cassandra, DHT technologies. storage and retrieval of information are all event-driven responsible by the EQ module.

5.2.2 Event Queue

Event Queue (EQ) module is responsible for event classification, event prioritization and event matching.

Firstly, EQ module provides isolated event queues customized for different event groups. Specifically, two groups of AFs could have completely different purposes or interests, therefore EQ classification allows to create multiple message queues where only AFs interested in the same category of events will be aware of the corresponding event queue.

Secondly, events generated may have to be processed with different priorities. Some of them are more urgent than the normal and regular ones. Also between two event queues, their priorities may be different. EQ prioritization allows AFs to set different priorities on the information they published. Based on the priority settings in the event queue, matching and delivery of them will be adjusted. EQ module can provide several pre-defined priority levels for both intra-queue and inter-queue prioritizations.

Third, events in queues will be listened and if a publishing event is found and matched by a registration event, information retrieval will be triggered.

5.2.3 Interface between IS and EQ Modules

EQ and IS modules are correlated. When an AF publishes information, not only an publishing event is translated and sent to EQ module, but also the information is indexed and stored simultaneously. Similarly, when an AF subscribes information, not only subscribing event is triggered and sent to EQ module, but also the information will be retrieved by IS module at the same time.

5.3 Summary

In summary, the general requirements for the information distribution module on each autonomic node are two sub-modules handling instant communications and asynchronous communications, respectively. For instant communications, node requirements are simple, in which signaling protocols have to be supported. With minimum efforts, reusing the existing GRASP is possible. For asynchronous communications, information distribution module requires event queue and information storage mechanism to be supported.

6. Protocol Specification (GRASP extension)

There are multiple ways to integrate the information distribution module. The principle we follow is to minimize modifications made to the current ANI.

We consider to use GRASP as an interface to access the information

distribution module. The main reason is that the current version of
GRASP is already an information distribution module for the cases of
P2P and flooding. In the following discussions, we introduce how to
complete the missing part.

6.1 Un-solicited Synchronization Message (A new GRASP Message)

In fragmentary CDDL, a Un-solicited Synchronization message follows
the pattern:

    unsolicited_synch-message = [M_UNSOLDSYNCH, session-id, objective]

A node MAY actively send a unicast Un-solicited Synchronization
message with the Synchronization data, to another node. This MAY be
sent to port GRASP_LISTEN_PORT at the destination address, which
might be obtained by GRASP Discovery or other possible ways. The
synchronization data are in the form of GRASP Option(s) for specific
synchronization objective(s).

6.2 Selective Flooding Option

In fragmentary CDDL, the selective flood follows the pattern:

    selective-flood-option = [O_SELECTIVE_FLOOD, +O_MATCH-CONDITION,
                              match-object, action]
    O_MATCH-CONDITION = [O_MATCH-CONDITION, Obj1, match-rule, Obj2]
       Obj1 = text
       match-rule = GREATER / LESS / WITHIN / CONTAIN
       Obj2 = text
    match-object = NEIGHBOR / SELF
    action = FORWARD / DROP

The selective flood option encapsulates a match-condition option
which represents the conditions regarding to continue or discontinue
flood the current message. For the match-condition option, the Obj1
and Obj2 are to objects that need to be compared. For example, the
Obj1 could be the role of the device and Obj2 could be "RSG". The
match rules between the two objects could be greater, less than,
within, or contain. The match-object represents of which Obj1 belongs
to, it could be the device itself or the neighbor(s) intended to be
flooded. The action means, when the match rule applies, the current
device just continues flood or discontinues.

6.3 Subscription Objective Option

In fragmentary CDDL, a Subscription Objective Option follows the
pattern:

```
subscription-objection-option = [SUBSCRIPTION, 2, 2, subobj]
objective-name = SUBSCRIPTION
objective-flags = 2
loop-count = 2
subobj = text
```

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a subscription to a specific object.

6.4 Un_Subscription Objective Option

In fragmentary CDDL, a Un_Subscribe Objective Option follows the pattern:

```
Unsubscribe-objection-option = [UNSUBSCRIB, 2, 2, unsubobj]
objective-name = SUBSCRIPTION
objective-flags = 2
loop-count = 2
unsubobj = text
```

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a un-subscription to a specific object.

6.5 Publishing Objective Option

In fragmentary CDDL, a Publish Objective Option follows the pattern:

```
publish-objection-option = [PUBLISH, 2, 2, pubobj] objective-name
= PUBLISH
objective-flags = 2
loop-count = 2
pubobj = text
```

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a publish of a specific object data.

Note that extended GRASP messages with new arguments inside here will trigger interactions/actions of the underlying information distribution module introduced in Section 5.

7. Security Considerations

   The distribution source authentication could be done at multiple
   layers:

      o Outer layer authentication: the GRASP communication is within
      ACP (Autonomic Control Plane,
      [I-D.ietf-anima-autonomic-control-plane]). This is the default
      GRASP behavior.

      o Inner layer authentication: the GRASP communication might not
      be within a protected channel, then there should be embedded
      protection in distribution information itself. Public key
      infrastructure might be involved in this case.

8. IANA Considerations

   TBD.

9.   References

9.1 Normative References

   [I-D.ietf-anima-grasp]
             Bormann, D., Carpenter, B., and B. Liu, "A Generic
             Autonomic Signaling Protocol (GRASP)", draft-ietf-
             animagrasp-15 (Standard Track), October 2017.

9.2 Informative References

   [RFC7575] Behringer, M., "Autonomic Networking: Definitions and
             Design Goals", RFC 7575, June 2015

   [I-D.ietf-anima-autonomic-control-plane]
             Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic
             Control Plane (ACP)", draft-behringer-anima-autonomic-
             control-plane-13, December 2017.

   [I-D.ietf-anima-stable-connectivity-10]
             Eckert, T., Behringer, M., "Using Autonomic Control Plane
             for Stable Connectivity of Network OAM", draft-ietf-anima-
             stable-connectivity-10, February 2018.

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

                anima-reference-model-05, October 2017.

   [I-D.du-anima-an-intent]
                Du, Z., Jiang, S., Nobre, J., Ciavaglia, L., and M.
                Behringer, "ANIMA Intent Policy and Format", draft-
                duanima-an-intent-05 (work in progress), February 2017.

   [I-D.ietf-anima-grasp-api]
                Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic
                Autonomic Signaling Protocol Application Program Interface
                (GRASP API)", draft-ietf-anima-grasp-api-00 (work in
                progress), December 2017.

   [I-D.carpenter-anima-grasp-bulk-02]
                Carpenter, B., Jiang, S., Liu, B., "Transferring Bulk Data
                over the GeneRic Autonomic Signaling Protocol (GRASP)",
                draft-carpenter-anima-grasp-bulk-02 (work in progress),
                June 30, 2018

   [3GPP.29.500]
                3GPP, "Technical Realization of Service Based
                Architecture", 3GPP TS 29.500 15.1.0, 09 2018

   [3GPP.23.501]
                3GPP, "System Architecture for the 5G System", 3GPP TS
                23.501 15.2.0, 6 2018,
                <http://www.3gpp.org/ftp/Specs/html-info/23501.htm>.

   [3GPP.23.502]
                3GPP, "Procedures for the 5G System", 3GPP TS 23.502
                15.2.0, 6 2018, <http://www.3gpp.org/ftp/Specs/html-
                info/23502.htm>.

   [5GAA.use.cases]
                White Paper "Toward fully connected vehicles: Edge
                computing for advanced automotive communications", 5GAA
                <http://5gaa.org/news/toward-fully-connected-vehicles-
                edge-computing-for-advanced-automotive-communications/>

Appendix A.

   GRASP includes flooding criteria together with the delivered
   information so that every node will process and act according to the
   criteria specified in the message. An example of extending GRASP with

selective criteria can be:

   o Matching condition: "Device role=IPRAN_RSG"

   o Matching objective: "Neighbors"

   o Action: "Forward"

This example means: only distributing the information to the
neighbors who are IPRAN_RSG.

Authors' Addresses

Bing Liu
Huawei Technologies
Q27, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: leo.liubing@huawei.com

Sheng Jiang
Huawei Technologies
Q27, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: jiangsheng@huawei.com

Xun Xiao
Munich Research Center
Huawei technologies
Riesstr. 25, 80992, Muenchen, Germany

Emails: xun.xiao@huawei.com

Artur Hecker
Munich Research Center
Huawei technologies
Riesstr. 25, 80992, Muenchen, Germany

Emails: artur.hecker@huawei.com

Zoran Despotovic

Munich Research Center
Huawei technologies
Riesstr. 25, 80992, Muenchen, Germany

Emails: zoran.despotovic@huawei.com

Appendix A   Real-world Use Cases of Information Distribution

The requirement analysis in Section 3 shows that generally
information distribution should be better of as an infrastructure
layer module, which provides to upper layer utilizations. In this
section, we review some use cases from the real-world where an
information distribution module with powerful functions do plays a
critical role there.

A.1 Service-Based Architecture (SBA) in 3GPP 5G

In addition to Internet, the telecommunication network (i.e. carrier
mobile wireless networks) is another world-wide networking system.
The architecture of the upcoming 5G mobile networks from 3GPP has
already been defined to follow a service-based architecture (SBA)
where any network function (NF) can be dynamically associated with
any other NF(s) when needed to compose a network service. Note that
one NF can simultaneously associate with multiple other NFs, instead
of being physically wired as in the previous generations of mobile
networks. NFs communicate with each other over service-based
interface (SBI), which is also standardized by 3GPP [3GPP.23.501].

In order to realize an SBA network system, detailed requirements are
further defined to specify how NFs should interact with each other
with information exchange over the SBI. We now list three
requirements that are related to information distribution here.

   1) NF Pub/Sub: Any NF should be able to expose its service status
   to the network and any NF should be able to subscribe the service
   status of an NF and get notified if the status is available. An
   concrete example is that a session management function (SMF) can
   subscribe the REGISTER notification from an access management
   function (AMF) if there is a new user entity trying to access the
   mobile network [3GPP.23.502].

   2) Network Exposure Function (NEF): A particular network function
   that is required to manage the event exposure and distributions.
   In specific, SBA requires such a functionality to register network
   events from the other NFs (e.g. AMF, SMF and so on), classify the
   events and properly handle event distributions accordingly in

terms of different criteria (e.g. priorities) [3GPP.23.502].

    3) Network Repository Function (NRF): A particular network
    function where all service status information is stored for the
    whole network. An SBA network system requires all NFs to be
    stateless so as to improve the resilience as well as agility of
    providing network services. Therefore, the information of the
    available NFs and the service status generated by those NFs will
    be globally stored in NRF as a repository of the system. This
    clearly implies storage capability that keeps the information in
    the network and provides those information when needed. A concrete
    example is that whenever a new NF comes up, it first of all
    registers itself at NRF with its profile. When a network service
    requires a certain NF, it first inquires NRF to retrieve the
    availability information and decides whether or not there is an
    available NF or a new NF must be instantiated [3GPP.23.502].

(Note: 3GPP CT might finally adopt HTTP2.0/JSON to be the protocol
communicating between NFs, but autonomic networks can also load
HTTP2.0 with in ACP.)

A.2 Vehicle-to-Everything

Carrier networks On-boarding services of vertical industries are also
one of some blooming topics that are heavily discussed. Connected car
is clearly one of the important scenarios interested in automotive
manufacturers, carriers and vendors. 5G Automotive Alliance – an
industry collaboration organization defines many promising use cases
where services from car industry should be supported by the 5G mobile
network. Here we list two examples as follows [5GAA.use.cases].

    1) Software/Firmware Update: Car manufacturers expect that the
    software/firmware of their car products can be remotely
    updated/upgraded via 5G network in future, instead of onsite
    visiting their 4S stores/dealers offline as nowadays. This
    requires the network to provide a mechanism for vehicles to
    receive the latest software updates during a certain period of
    time. In order to run such a service for a car manufacturer, the
    network shall not be just like a network pipe anymore. Instead,
    information data have to be stored in the network, and delivered
    in a publishing/subscribing fashion. For example, the latest
    release of a software will be first distributed and stored at the
    access edges of the mobile network, after that, the updates can be
    pushed by the car manufacturer or pulled by the car owner as
    needed.

    2) Real-time HD Maps: Autonomous driving clearly requires much
    finer details of road maps. Finer details not only include the

details of just static road and streets, but also real-time
information on the road as well as the driving area for both local
urgent situations and intelligent driving scheduling. This asks
for situational awareness at critical road segments in cases of
changing road conditions. Clearly, a huge amount of traffic data
that are real-time collected will have to be stored and shared
across the network. This clearly requires the storage capability,
data synchronization and event notifications in urgent cases from
the network, which are still missing at the infrastructure layer.

A.3 Summary

Through the general analysis and the concrete examples from the real-
world, we realize that the ways information are exchanged in the
coming new scenarios are not just short and instant anymore. More
advanced as well as diverse information distribution capabilities are
required and should be generically supported from the infrastructure
layer. Upper layer applications (e.g. ASAs in ANIMA) access and
utilize such a unified mechanism for their own services.