ANIMA WG M. Behringer, Ed.
Internet-Draft Cisco Systems
Intended status: Standards Track T. Eckert
Expires: September 28, 2017 Huawei
S. Bjarnason
Arbor Networks
March 27, 2017

An Autonomic Control Plane
draft-ietf-anima-autonomic-control-plane-06

Abstract

Autonomic functions need a control plane to communicate, which depends on some addressing and routing. This Autonomic Control Plane should ideally be self-managing, and as independent as possible of configuration. This document defines an "Autonomic Control Plane", with the primary use as a control plane for autonomic functions. It also serves as a "virtual out of band channel" for OAM communications over a network that is not configured, or mis-configured.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

Autonomic functions need a stable and robust infrastructure to communicate on. This infrastructure should be as robust as possible, and it should be re-usable by all autonomic functions. [RFC7575] calls it the "Autonomic Control Plane". This document defines the Autonomic Control Plane.

Today, the management and control plane of networks typically runs in the global routing table, which is dependent on correct configuration and routing. Misconfigurations or routing problems can therefore disrupt management and control channels. Traditionally, an out of band network has been used to recover from such problems, or personnel is sent on site to access devices through console ports (craft ports). However, both options are operationally expensive.

In increasingly automated networks either controllers or distributed autonomic service agents in the network require a control plane which

is independent of the network they manage, to avoid impacting their own operations.

This document describes options for a self-forming, self-managing and self-protecting "Autonomic Control Plane" (ACP) which is inband on the network, yet as independent as possible of configuration, addressing and routing problems (for details how this achieved, see Section 5). It therefore remains operational even in the presence of configuration errors, addressing or routing issues, or where policy could inadvertently affect control plane connectivity. The Autonomic Control Plane serves several purposes at the same time:

o Autonomic functions communicate over the ACP. The ACP therefore supports directly Autonomic Networking functions, as described in [I-D.ietf-anima-reference-model]. For example, GRASP [I-D.ietf-anima-grasp] can run securely inside the ACP.

o An operator can use it to log into remote devices, even if the data plane is misconfigured or unconfigured.

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

This document describes some use cases for the ACP in Section 2, it defines the requirements in Section 3, Section 4 gives an overview how an Autonomic Control Plane is constructed, and in Section 5 the detailed process is explained. Section 6 explains how non-autonomic nodes and networks can be integrated, and Section 5.5 the first channel types for the ACP.

The document "Autonomic Network Stable Connectivity" [I-D.ietf-anima-stable-connectivity] describes how the ACP can be used to provide stable connectivity for OAM applications. It also explains on how existing management solutions can leverage the ACP in parallel with traditional management models, when to use the ACP versus the data plane, how to integrate IPv4 based management, etc.

2. Use Cases for an Autonomic Control Plane

2.1. An Infrastructure for Autonomic Functions

Autonomic Functions need a stable infrastructure to run on, and all autonomic functions should use the same infrastructure to minimise the complexity of the network. This way, there is only need for a

single discovery mechanism, a single security mechanism, and other processes that distributed functions require.

2.2. Secure Bootstrap over an Unconfigured Network

Today, bootstrapping a new device typically requires all devices between a controlling node (such as an SDN controller) and the new device to be completely and correctly addressed, configured and secured. Therefore, bootstrapping a network happens in layers around the controller. 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 between.

With the ACP, secure bootstrap of new devices can happen without requiring any configuration on the network. A new device can automatically be bootstrapped in a secure fashion and be deployed with a domain certificate. This does not require any configuration on intermediate nodes, because they can communicate through the ACP.

2.3. Data Plane Independent Permanent Reachability

Today, most critical control plane protocols and network management protocols are running in the data plane (global routing table) of the network. This leads to undesirable dependencies between control and management plane on one side and the data plane on the other: Only if the data plane is operational, will the other planes work as expected.

Data plane connectivity can be affected by errors and faults, for example certain AAA misconfigurations can lock an administrator out of a device; routing or addressing issues can make a device unreachable; shutting down interfaces over which a current management session is running can lock an admin irreversibly out of the device. Traditionally only console access can help recover from such issues.

Data plane dependencies also affect NOC/SDN controller applications: Certain network changes are today hard to operate, because the change itself may affect reachability of the devices. Examples are address or mask changes, routing changes, or security policies. Today such changes require precise hop-by-hop planning.

The ACP provides reachability that is largely independent of the data plane, which allows 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 devices regardless of their configuration or global routing table.

- For control plane protocols, the ACP allows their operation even when the data plane is temporarily faulty, or during transitional events, such as routing changes, which may affect the control plane at least temporarily. This is specifically important for autonomic service agents, which could affect data plane connectivity.

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

3. Requirements

The Autonomic Control Plane has the following requirements:

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

ACP2: The ACP MUST have a separate address space from the data plane. Reason: traceability, debug-ability, separation from data plane, security (can block easily at edge).

ACP3: The ACP MUST use autonomically managed address space. Reason: easy bootstrap and setup ("autonomic"); robustness (admin can't mess things up so easily). This document suggests to use ULA addressing for this purpose.

ACP4: The ACP MUST be generic. Usable by all the functions and protocols of the AN infrastructure. It MUST NOT be tied to a particular protocol.

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

The default mode of operation of the ACP is 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 over non-autonomic nodes, for example to link autonomic nodes over the general Internet. This is possible, but then has a dependency on routing over the non-autonomic hops.

4. Overview

The Autonomic Control Plane is constructed in the following way (for details, see Section 5):

1. An autonomic 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 adjacent nodes in the same domain.

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

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

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

6. Each node runs a lightweight routing protocol, to announce reachability of the virtual addresses inside the ACP.

Note:

- o Non-autonomic NMS systems or controllers have to be manually connected into the ACP.

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

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

The following figure illustrates the ACP.

```
           autonomic node 1                  autonomic node 2
          ...................               ...................
   secure .                 .   secure      .                 .  secure
   tunnel :  +-----------+  :   tunnel      :  +-----------+  :  tunnel
   ..--------| ACP VRF   |-----------------------| ACP VRF   |---------..
          : / \         / \   <--routing-->   / \         / \ :
          : \ /         \ /                   \ /         \ / :
   ..--------|  virtual  |-----------------------|  virtual  |---------..
          :  | interface |  :               :  | interface |  :
          :  +-----------+  :               :  +-----------+  :
          :                 :               :                 :
          :   data plane    :...............:   data plane    :
          :                 :    link       :                 :
          :.................:               :.................:
```

Figure 1

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

5. Self-Creation of an Autonomic Control Plane

This section describes the steps to set up an Autonomic Control Plane, and highlights the key properties which make it "indestructible" against many inadvert changes to the data plane, for example caused by misconfigurations.

5.1. Preconditions

An autonomic node can be a router, switch, controller, NMS host, or any other IP device. We assume an autonomic node has a globally unique domain certificate (LDevID), as well as an adjacency table.

5.1.1. Domain Certificate with ANIMA information

To establish an ACP securely, an Autnomic Node MUST have a globally unique domain certificate (LDevID), with which it can cryptographically assert its membership of the domain. The document [I-D.ietf-anima-bootstrapping-keyinfra] describes how a domain certificate can be automatically and securely derived from a vendor specific Unique Device Identifier (UDI) or IDevID certificate.

The domain certificate (LDevID) of an autonomic node MUST contain ANIMA specific information, specifically the domain name, and its ACP

address with the zone-ID set to zero. This information MUST be encoded in the LDevID in the subjectAltName / rfc822Name field in the following way:

anima.acp+<ACP address>@<domain>

Example:

anima.acp+FD99:B02D:8EC3:0:200:0:6400:1@example.com

The ACP address MUST be specified in its canonical form, as specified in [RFC5952], to allow for easy textual comparisons.

The particular subjectAlName / rfc822Name encoding is choosen for several reasons:

- We want to permit the reuse of the ANIMA LDevID in other uses beside the ACP as appropriate, eg: there are a wide number of pre-existing data-plane security mechanisms where re-using the ACP certificate could help to further automate security.

- We therefore want to make sure that ACP elements in the LDevID will not cause incompatibilities with any pre-existing ASN.1 code potentially in use in those other pre-existing SW systems.

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

- We also want to make sure that the ACP information will not be misinterpreted by any such pre-existing code interpreting the LDevID, or if it is misinterpreted, that the impact is benign.

- Using an IP address format encoding could result in non-benign misinterpretation of the ACP information.

- At minimum, we need to encode both the AN domain name and the non-domain name derived part of the ACP, so there are not many alternatives with pre-existing fields where those two elements could be encoded.

- rfc822Name encoding allows to be quite flexible. We choose to encode the full ACP address AND the domain name, so that it is easier to examine/use.

- The format of the rfc822Name is choosen so that an operator can set up a mailbox called   anima.acp@<domain> that would receive emails sent towards the rfc822Name of any AN device inside a

domain. This is possible because components behind a plus symbol are considered part of a single mailbox.

- 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.

The bootstrap process defined in [I-D.ietf-anima-bootstrapping-keyinfra] MUST in an ANIMA network pass on ACP address and domain to a new node, such that the new node can add this to its enrolment request.

The Certificate Authority in an ANIMA network MUST honor these parameters, and create the respective subjectAltName / rfc822Name in the certificate.

ANIMA nodes can therefore find ACP address and domain using this field in the domain certificate, both for themselves, as well as for other nodes.

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

5.1.2. AN Adjacency Table

To know to which nodes to establish an ACP channel, every autonomic node maintains an adjacency table. The adjacency table contains information about adjacent autonomic nodes, at a minimum: node-ID, IP address, domain, certificate. An autonomic device 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 autonomic device is not directly adjacent, the information in the adjacency table can be supplemented by configuration. For example, the node-ID and IP address could be configured.

The adjacency table MAY contain information about the validity and trust of the adjacent autonomic node's certificate. However, subsequent steps MUST always start with authenticating the peer.

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

5.2. Neighbor discovery

5.2.1. L2 topology considerations

```
       ANrtr1 ------ ANswitch1 --- ANswitch2 ------- ANrtr2
                 .../   \                   \  ...
       ANrtrM ------     \                   ------- ANrtrN
                          ANswitchM ...
```

Figure 2

Consider a large L2 LAN with ANrtr1...ANrtrN connected via some topology of L2 switches (eg: in a large enterprise campus or IoT environment using large L2 LANs). If the discovery protocol used for the ACP is operating at the subnet level, every AN router will see all other AN routers on the LAN as neighbors and a full mesh of ACP channels will be built. If some or all of the AN switches are autonomic with the same discovery protocol, then the full mesh would include those switches as well.

A full mesh of ACP connections like this can creates fundamental challenges. The number of security associations of the secure channel protocols will not scale arbitrarily, especially when they leverage platform accelerated encryption/decryption. Likewise, any other ACP operations needs to scale to the number of direct ACP neigbors. An AN router with just 4 interfaces might be deployed into a LAN with hundreds of neighbors connected via switches. Introducing such a new unpredictable scaling factor requirement makes it harder to support the ACP on arbitrary platforms and in arbitrary deployments.

Predictable scaling requirements for ACP neighbors can most easily be achieved if in topologies like these, AN capable L2 switches can ensure that discovery messages terminate on them so that neighboring AN routers and switches will only find the physcially connected AN 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 physcial 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 AN capable, and ANswitch2 is not AN capable. The desired ACP topology is therefore that ANrtr1 and ANrtrM only have an ACP connetion 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.

5.2.2. CDP/LLDP/mDNS considerations

LLDP (and Cisco's 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-ANIMA switches.

mDNS operates at the subnet level, and is also used on L2 switches. The authors of this document are not aware of mDNS implementation that terminate their messages on L2 ports instead of the subnet level. If mDNS was used as the ACP discovery mechanism on an ACP capable L2 switch, then this would be necessary to implement. It is likely that termination of mDNS messages could only be applied to all mDNS messages from a port, which would then make it necessary to software forward any non-ACP related mDNS messages to maintain prior non-ACP mDNS functionality. With low performance of software forwarding in many L2 switches, this could easily make the ACP unsupportable on such L2 switches.

5.2.3. Discovery with GRASP

In conclusion for the above described considerations, the ACP uses "insecure" instances of GRASP for discovery of ACP neighbors because it can easily be set up to match the requiremetns without affecting other uses of the protocol.

The name of the GRASP objective to announce/discover the capability of a neighbor to run the ACP is "ACP". Section 3.5.2.2 of [I-D.ietf-anima-grasp] describes the instance of GRASP to be used for this purpose: "DULL" (Discovery Unsolicited Link Local). The precise GRASP objective to be used is specified in Section 3 of [I-D.carpenter-anima-ani-objectives].

As explained above, in an ACP enabled L2 switch, each of these instances would actually need to be per-L2-port. The result of the discovery is the IPv6 link-local address of the neighbor. It is stored in the AN Adjacency Table, see Section 5.1.2 which then drives the further building of the ACP to that neighbor.

For example, 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. This is easily achieved by extracting native GRASP multicast messages by their MAC multicast destination address. None of the other type of GRASP

instances (eg: as used inside the ACP) use GRASP messages that would be affected by such extraction, because all other GRASP messages have encrypted encapsulations.

5.2.4. Discovery and BRSKY

Before a node has a domain certificate, it can not participate in the ACP and therefore does also not try to discover an ACP neighbor. Instead, it uses the discovery mechanism described in [I-D.ietf-anima-grasp] to discover a bootstrap proxy. Currently, that document describes mDNS as the protocol of choice for that discovery. In the context of above topology example, ANrtr1 might therefore discover and choose any ANrtr or ANswitch on the LAN that is already part of the autonomic domain - instead of the closest one which is ANswitch1. This choice of bootstrap proxy does not impact in the later building of the ACP on ANrtr1 and is therefore not a concern for the ACP.

Once a device has its domain certificate, it will start announcing itself via GRASP as ACP capable.

When an autonomic device is a registrar, it will announce the registrar function via GRASP in the ACP as the "6JOIN" objective. An AN device that is a registrar or learns about one or more reachable registrars via this GRASP/ACP announcements will announce itself as a boostrap proxy via mDNS. See [I-D.richardson-anima-6join-discovery] for more details.

5.3. Candidate ACP Neighbor Selection

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

The ACP is by default established exclusively between nodes in the same domain.

Intent can change this default behaviour. Since Intent is transported over the ACP, the first ACP connection a node establishes is always following the default behaviour. The precise format for this Intent needs to be defined outside this document. Example Intent policies which need to be supported include:

o The ACP should be built between all sub-domains for a given parent domain. For example: For domain "example.com", nodes of "example.com", "access.example.com", "core.example.com" and "city.core.example.com" should all establish one single ACP.

o Two domains should build one single ACP between themselves, for example "example1.com" should establish the ACP also with nodes from "example2.com". For this case, the two domains must be able to validate their trust, typically by cross-signing their certificate infrastructure.

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.

5.4. Channel Selection

To avoid attacks, initial discovery of candidate ACP peers can not include any non-protected negotiation. To avoid re-inventing and validating security association mechanisms, the next step after discoving 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 autonomic devices, it is unclear whether it is feasible to even decide on a single MTI (mandatory to implement) security association protocol across all autonomic devices.

From the use-cases it is clear that not all type of autonomic devices 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 use-cases), but low-end in-ceiling L2 switches may only want to support MacSec because that is also supported in HW, and only a more flexible garteway device may need to support both of these mechanisms and potentially more.

To support these requirements, and make ACP channel negotiation also easily extensible, the secure channel selection between Alice and Bob is a potentially two stage process. In the first stage, Alice and Bob directly try to establish a secure channel using the security-association and channel protocols they support. One or more of these protocols may simply be protocols not directly resulting in an ACP channel, but instead establishing a secure negotiation channel through which the final secure channel protocol is decided. If both Alice and Bob support such a negotiation step, then this secured negotiation channel is the first step, and the secure channel protocol is the second step.

If Alice supports multiple security association protocols in the first step, it is a matter of Alices local policy to determine the

order in which she will try to build the connection to Bob. To support multiple security association protocols, Alice must be able to simultaneously act as a responder in parallel for all of them - so that she can respond to any order in which Bob wants to prefer building the security association.

When ACP setup between Alice and Bob results in the first secure association to be established, the peer with the higher Device-ID in the certificate will stop building new security associations. The peer with the lower certificate Device-ID is now responsible to continue building its most preferred security association and to tear down all but that most preferred one - unless the secure association is one of a negotation protocols whose rules superceed this.

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.

5.5. Security Association protocols

The following sections define the security association protocols that we consider to be important and feasible to specify in this document. In all cases, the mutual authentication is done via the autonomic domain certificate of the peer as follows - unless superceeded by Intent:

o The certificate is valid as proven by the security associations protocol exchanges.

o If the certificate is included in a Certificate Revocation List (CRL), the connection attempt is aborted and an error logged. [EDNOTE: Do we want OCSP instead of CRL?] [EDNOTE: Distribution of the CRL, and handling of CRL timeouts during network partition needs to be discussed in more detail.]

o The peers certificate is signed by the same CA as the devices domain certificate.

o The peers OU (Organizational Unit) field in the certificates Subject is the same as in the devices certificate.

5.5.1. ACP via IPsec

To run ACP via IPsec transport mode, no further IANA assignments/ definitions are required. All autonomic devices suppoting IPsec MUST support IPsec security setup via IKEv2, transport mode encapsulation via the device and peer link-local IPv6 addresses and AES256 encryption.

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

5.5.2. ACP via GRE/IPsec

In network devices it is often easier to provide virtual interfaces on top of GRE encapsulation than natively on top of a simple IPsec association. On those devices it may be necessary to run the ACP secure channel on top of a GRE connection protected by the IPsec association. The requirements for the IPsec association are the same as described above, but instead of directly carrying the ACP IPv6 packets, the payload is an ACP IPv6 packet inside GRE/IPv6.

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 41 (IPv6) (because native IPsec is required to be supported, see below).

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

5.5.3. 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] an IANA assigned port [TBD] is used. All autonomic devices supporting ACP via dTLS must use AES256 encryption.

There is no additional session setup or other security association other than dTLS. As soon as the dTLS session is functional, the ACP peers will exchange ACP IPv6 packets as the payload of the dTLS transport connecetion. Any dTLS defined security association mechanisms such as re-keying are used as they would be for any transport application relying solely on dTLS.

5.5.4. GRASP/TLS negotiation

To explicitly allow negotiation of the ACP channel protocol, GRASP over a TLS connection using the GRASP_LISTEN_PORT and the devices 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 initially negotiate the channel mechanism to use. Bob and Alice each have a list of channel mehanisms they support, sorted by preference. They negotiate the best mechansm supported by both of them. In the absence of Intent, the mechanism choosen is the best one for the device with the lower Device-ID.

After agreeing on a channel mechanism, Alice and Bob start the selected Channel protocol. The GRASP/TLS connection can be kept alive or timed out as long as the seelected channel protocol has a secure association between Alice and Bob. When it terminates, it needs to be re-negotiated via GRASP/TLS.

Negotiation of a channel type may require IANA assignments of code points. See IANA Considerations (Section 11) for the formal definition of those code points.

The exact negotiation steps in GRASP to achieve this outcome.

5.5.5. ACP Security Profiles

A baseline autonomic device MUST support IPsec and SHOULD support GRASP/TLS and dTLS. A constrained autonomic device MUST support dTLS.

The MTU for ACP secure channels must be derived locally from the underlying link MTU minus the security encapsulation overhead. Given how ACP channels are built across layer2 connections only, the probability for MTU mismatch is low. For additional reliability, applications to be runa cross the ACP should only assume to have minimum MTU available (1280).

Autonomic devices need to specify in documentation the set of secure ACP mechanisms they suppport.

5.6. GRASP instance details

Received GRASP packets are assigned to an instance of GRASP by the context they are received on:

o GRASP packets received on an ACP (virtual) interfaces are assigned to the ACP instance of GRASP

o GRASP/UDP packets received on L2 interfaces/ports where the device is willing to run ACP are assigned to a DULL instance of GRASP for that interface/port.

- o GRASP packets received inside a TLS connection established for GRASP/TLS ACP negotiation are assigned to a separate instance of GRASP for that negotiation.

5.7. Context Separation

The ACP is in a separate context from the normal data plane of the device. This context includes the ACP channels IPv6 forwarding and routing as well as any required higher layer ACP functions.

In classical network device platforms, a dedicated so called "Virtual routing and forwarding instance" (VRF) is one logical implementation option for the ACP. If possible by the platform SW 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 device. As much as possible it should be protected from being modified unintentionally by data plane configuration.

Context separation improves security, because the ACP is not reachable from the global routing table. Also, configuration errors from the data plane setup do not affect the ACP.

5.8. 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 internal network wide valid addresses and routing. Each autonomic node must create a virtual interface with a network wide unique address inside the ACP context mentioned in Section 5.7. This address may be used also in other virtual contexts.

With the algorithm introduced here, all autonomic devices in the same domain have the same /48 prefix. Conversely, global IDs from different domains are unlikely to clash, such that two networks can be merged, as long as the policy allows that merge. See also Section 7 for a discussion on merging domains.

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

5.8.1. 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.

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 loopback interfaces of autonomic nodes carry a routable address; all other interfaces exclusively use IPv6 link local for autonomic functions. The usage of IPv6 link local addressing is discussed in [RFC7404].

o Use-ULA: For loopback interfaces of autonomic nodes, we use Unique Local Addresses (ULA), as specified in [RFC4193]. An alternative scheme was discussed, using assigned ULA addressing. The consensus was to use ULA-random [[RFC4193] with L=1], because it was deemed to be sufficient.

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].

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

5.8.2. The ACP Addressing Base Scheme

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

```
  8      40          3                     77
+--+--------------+------+------------------------------------------+
|FD| hash(domain) | Type |             (sub-scheme)                 |
+--+--------------+------+------------------------------------------+
```

Figure 3: ACP Addressing Base Scheme

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

- o "FD" identifies a locally defined ULA address.

- o The ULA "global ID" is set here to be a hash of the domain name, which results in a pseudo-random 40 bit value. It is calculated as the first 40 bits of the SHA256 hash of the domain name, in the example "example.com".

- o Type: This field allows different address sub-schemes in the future. The goal is to start with a single sub-schemes, but to allow for extensions later if and when required. This addresses the "upgradability" requirement. Assignment of types for this field should be maintained by IANA.

5.8.3. ACP Addressing Sub-Scheme

The sub-scheme defined here is defined by the Type value 0 (zero) in the base scheme.

```
          51                 13                    63            1
+------------------------+---------+----------------------------+---+
|    (base scheme)       | Zone ID |         Device ID          | V |
+------------------------+---------+----------------------------+---+
```

Figure 4: ACP Addressing Sub-Scheme

The fields are defined as follows: [Editor's note: The lengths of the fields is for discussion.]

- o Zone ID: If set to all zero bits: The device 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 Section 5.8.4 on how this field is used in detail.

- o Device ID: A unique value for each device.

- o V: Virtualization bit: 0: autonomic node base system; 1: a virtual context on an autonomic node.

The device ID is derived as follows: In an Autonomic Network, a registrar is enrolling new devices. As part of the enrolment process the registrar assigns a number to the device, which is unique for this registrar, but not necessarily unique in the domain. The 64 bit device ID is then composed as:

- o 48 bit: Registrar ID, a number unique inside the domain that identifies the registrar which assigned the name to the device. A MAC address of the registrar can be used for this purpose.
- o 15 bit: Device number, a number which is unique for a given registrar, to identify the device. This can be a sequentially assigned number.

The "device ID" itself is unique in a domain (i.e., the Zone-ID is not required for uniqueness). Therefore, a device can be addressed either as part of a flat hierarchy (zone ID = 0), or with an aggregation scheme (any other zone ID). A address with zone-ID = 0 is an identifier, with another zone-ID as a locator. See Section 5.8.4 for a description of the zone bits.

This addressing sub-scheme allows the direct addressing of specific virtual containers / VMs on an autonomic node. An increasing number of hardware platforms have a distributed architecture, with a base OS for the node itself, and the support for hardware blades with potentially different OSs. The VMs on the blades could be considered as separate autonomic nodes, in which case it would make sense to be able to address them directly. Autonomic Service Agents (ASAs) could be instantiated in either the base OS, or one of the VMs on a blade. This addressing scheme allows for the easy separation of the hardware context.

The location of the V bit(s) at the end of the address allows to announce a single prefix for each autonomic node, while having separate virtual contexts addressable directly.

[EDNOTE: various suggestions from mcr in his mail from 30 Nov 2016 to be considered (https://mailarchive.ietf.org/arch/msg/anima/nZpEphrTqDCBdzsKMpaIn2gsIzI).]

5.8.4. Usage of the Zone Field

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

Zone = zero is the default addressing scheme in an autonomic domain. Every autonomic node MUST respond to its ACP address with zone=0. Used on its own this leads to a non-hierarchical address scheme,

which is suitable for networks up to a certain size. In this case, the addresses primarily act as identifiers for the nodes, and aggregation is not possible.

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

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

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

Note: Another way to introduce hierarchy is to use sub-domains in the naming scheme. The node names "node17.subdomainA.example.com" and "node4.subdomainB.example.com" would automatically lead to different ULA prefixes, which can be used to introduce a routing hierarchy in the network, assuming that the subdomains are aligned with routing areas.

5.8.5. Other ACP Addressing Sub-Schemes

Other ACP addressing sub-schemes can be defined if and when required. IANA will assign a new "type" for each new addressing sub-scheme.

5.9. Routing in the ACP

Once ULA address are set up all autonomic entities should run a routing protocol within the autonomic control plane context. This routing protocol distributes the ULA created in the previous section for reachability. The use of the autonomic control plane specific context eliminates the probable clash with the global routing table and also secures the ACP from interference from the configuration mismatch or incorrect routing updates.

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

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

The routing protocol inside the ACP is RPL ([RFC6550]) with the following profile. See Appendix A for more details on the choice of RPL.

5.9.1. RPL Profile for the ACP

- RPL Mode of Operations (MOP): mode 3 "Storing Mode of Operations with multicast support". Implementations should support also other modes. Note: Root indicates mode in DIO flow.
- Objective Function (OF): Use OF0 [RFC6552]. No use of metric containers, Default RPLInstanceID = 0.
  * stretch_rank: none provided ("not stretched").
  * rank_factor: Derived from link speed: <= 100Mbps: LOW_SPEED_FACTOR(5), else HIGH_SPEED_FACTOR(1)
- Trickle: Not used; Data Path Validation: Not used.
- Proactive, aggressive DAO state maintenance:
  * Use K-flag in unsolicited DAO indicating change from previous information (to require DAO-ACK).
  * Retry such DAO DAO-RETRIES(3) times with DAO-ACK_TIME_OUT(256ms) in between.
- Administrative Preference ([RFC6552], 3.2.6 - to become root): Indicated in DODAGPreference field of DIO message.
  * Explicit configured "root": 0b100
  * Registrar (Default): 0b011
  * AN-connect (non registrar): 0b010
  * Default: 0b001.

The RPL root can create additional RPL instances with other OF and metrics as desired, eg: via intent.

5.10. General ACP Considerations

In order to be independent of configured link addresses, channels SHOULD use IPv6 link local addresses between adjacent neighbors wherever possible. This way, the ACP tunnels are independent of correct network wide routing.

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 5.9.

If two nodes are connected via several links, the ACP SHOULD be established on 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 devices, because state needs to be kept per ACP channel.

6. Workarounds for Non-Autonomic Nodes

6.1. Connecting a Non-Autonomic 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 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-autonomic NMS system does not have access to the ACP by default, just like any other external device.

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. On an adjacent autonomic node with ACP, the interface with the NMS host can be configured as "ACP Connect". In this case, all devices on this port, including the NMS host, is entirely and exclusively inside the ACP. It would likely require a second interface outside the ACP for connections between the NMS host and administrators, or Internet based services. This mode of connecting an NMS host has security consequences: All systems and processes connected to this implicitly trusted "ACP Connect" interface have access to all autonomic nodes on the entire ACP, without further authentication. Thus, this connection must be physically controlled.

The non-autonomic NMS host must be routed in the ACP. This involves two parts: 1) the NMS host must point default to the AN device for the ULA prefix used inside the ACP, and 2) the prefix used between AN node and NMS host must be announced into the ACP, and distributed there.

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

If an NMS host is autonomic itself, it negotiates access to the ACP with its neighbor, like any other autonomic node.

6.2. ACP through Non-Autonomic L3 Clouds

Not all devices in a network may be autonomic. If non-autonomic Layer-2 devices are between autonomic nodes, the communications described in this document should work, since it is IP based. However, non-autonomic Layer-3 devices do not forward link local autonomic messages, and thus break the Autonomic Control Plane.

One workaround is to manually configure IP tunnels between autonomic nodes across a non-autonomic Layer-3 cloud. The tunnels are represented on each autonomic node as virtual interfaces, and all autonomic transactions work across such tunnels.

Such manually configured tunnels are less "indestructible" than an automatically created ACP based on link local addressing, since they depend on correct data plane operations, such as routing and addressing.

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

7. 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 devices.

- If an existing device 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 device 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. Devices authenticate each other using the domain certificates to establish the ACP locally. Addressing inside the ACP remains unchanged, and the routing protocol inside both parts of the ACP will lead to two working (although partitioned) ACPs.

There are few central dependencies: A certificate revocation list (CRL) may not be available during a network partition; a suitable policy to not immediately disconnect neighbors when no CRL is available can address this issue. Also, a registrar or Certificate Authority might not be available during a partition. This may delay renewal of certificates that are to expire in the future, and it may prevent the enrolment of new devices during the partition.

After a network partition, a re-merge will just establish the previous status, certificates can be renewed, the CRL is available, and new devices can be enrolled everywhere. Since all devices 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 5.8).

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 whicht the ACP is running. Especially if bringing down the ACP is known to disconnect the operator from the device. 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).

8. Self-Protection Properties

As explained in Section 5, the ACP is based on secure channels built between devices that have mutually authenticated each other with their domain certificates. The channels themselves are protected using standard encryption technologies like DTLS or IPsec which provide additional authentication during channel establishment, data integrity and data confidentiality protection of data inside the ACP and in addition, provide replay protection.

An attacker will therefore 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 remaining attack vector would be to attack the underlying AN protocols themselves, either via directed attacks or by denial-of-service attacks. However, as the ACP is built using link-local IPv6 address, remote attacks are impossible. The ULA addresses are only reachable inside the ACP context, therefore unreachable from the data plane. Also, the ACP protocols should be implemented to be attack resistant and not consume unnecessary resources even while under attack.

9. 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 devices, 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 device 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 6.1 for more details on how to connect an NMS host into the ACP.

10. Security Considerations

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

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

- The usage of domain certificates depends on a valid supporting PKI infrastructure. If the chain of trust of this PKI infrastructure is compromised, the security of the ACP is also compromised. This is typically under the control of the network administrator.

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

There is no prevention of source-address spoofing inside the ACP. This implies that if an attacker gains access to the ACP, (s)he can spoof all addresses inside the ACP and fake messages from any other device.

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.

11. IANA Considerations

Section 5.5.3 describes ACP over dTLS, which requires a well-known UDP port. We request IANA to assign this UDP port for 'ACP over dTLS'.

Section 5.5.4 describes an option for the channel negotiation, the 'ACP channel type'. We request IANA to create a registry for 'ACP channel type'.

The ACP channel type is a 8-bit unsigned integer. This document only assigns the first value.

Number	Channel Type	RFC
0	GRE tunnel protected with IPsec transport mode	this document
1-255	reserved for future channel types	

Section 5.8.2 describes a 'type' field in the base addressing scheme. We request IANA to create a registry for the 'ACP addressing scheme type', with the following initial values:

Number	Address Type (sub-scheme)	RFC
0	Default address sub-scheme	this document
7	Reserved for private use sub-scheme	

12. 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, Ravi Kumar Vadapalli.

Special thanks to Pascal Thubert to provide the details for the recommendations of the RPL profile to use in the ACP

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

13. Change log [RFC Editor: Please remove]

13.1. Initial version

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

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

Initial version of the anima document; only minor edits.

13.3. 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.

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

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

- Added several references to GDNP as a candidate protocol.

- Added a discussion on network split and merge. Although, this should probably go into the certificate management story longer term.

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

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

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

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

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

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

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

- Added requirement section.

- Changed focus: more focus on autonomic functions, not only virtual out of band. This goes a bit throughout the document, starting with a changed abstract and intro.

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

No changes; re-submitted as WG document.

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

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

- Moved the data-plane ACP out of the main document, into an appendix. The focus is now the virtually separated ACP, since it has significant advantages, and isn't much harder to do.

- Changed the self-creation algorithm: Part of the initial steps go into the reference document. This document now assumes an adjacency table, and domain certificate. How those get onto the device is outside scope for this document.

- Created a new section 6 "workarounds for non-autonomic nodes", and put the previous controller section (5.9) into this new section. Now, section 5 is "autonomic only", and section 6 explains what to do with non-autonomic stuff. Much cleaner now.

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

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

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

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

- Updated links, lots of small edits.

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

- Added explicitly text for the ACP channel negotiation.

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

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

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

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

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

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

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

- Included information on how ANIMA information must be encoded in the domain certificate in Section 5.1.

- Editorial changes, updated draft references, etc.

13.10. 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).

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

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

- Section 5.8.1 (addressing): State that addresses in the ACP are permanent, and do not support temporary addresses as defined in RFC4941.

- Modified Section 5.2.3 to point to the GRASP objective defined in [I-D.carpenter-anima-ani-objectives]. (and added that reference)

- Section 5.8.2: changed from MD5 for calculating the first 40 bits to SHA256; reason is MD5 should not be used any more.

- Added address sub-scheme to the IANA section.

- Made the routing section more prescriptive.

- Clarified in Section 6.1 the ACP Connect port, and defined that term "ACP Connect".

- Section 6.2: Added some thoughts (from mcr) on how traversing a L3 cloud could be automated.

- Added a CRL check in Section 5.5.

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

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

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

- Added proposed RPL profile.

- detailed dTLS profile - dTLS with any additional negotiation/ signaling channel.

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

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

14. References

[I-D.carpenter-anima-ani-objectives]
Carpenter, B. and B. Liu, "Technical Objective Formats for the Autonomic Network Infrastructure", draft-carpenter-anima-ani-objectives-01 (work in progress), February 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-05 (work in progress), March 2017.

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

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

[I-D.ietf-anima-stable-connectivity]
Eckert, T. and M. Behringer, "Using Autonomic Control Plane for Stable Connectivity of Network OAM", draft-ietf-anima-stable-connectivity-02 (work in progress), February 2017.

[I-D.richardson-anima-6join-discovery]
Richardson, M., "GRASP discovery of Registrar by Join Assistant", draft-richardson-anima-6join-discovery-00 (work in progress), October 2016.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <http://www.rfc-editor.org/info/rfc4122>.

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

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

[RFC5082] Gill, V., Heasley, J., Meyer, D., Savola, P., Ed., and C. Pignataro, "The Generalized TTL Security Mechanism (GTSM)", RFC 5082, DOI 10.17487/RFC5082, October 2007, <http://www.rfc-editor.org/info/rfc5082>.

[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, <http://www.rfc-editor.org/info/rfc5280>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc6552>.

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

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

[RFC7404] Behringer, M. and E. Vyncke, "Using Only Link-Local Addressing inside an IPv6 Network", RFC 7404, DOI 10.17487/RFC7404, November 2014, <http://www.rfc-editor.org/info/rfc7404>.

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

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

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

Appendix A. Background on the choice of routing protocol

In a pre-standard implementation, the "IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL, [RFC6550] was chosen. This Appendix explains the reasoning behind that decision.

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 synchronisation.

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 behaviour.

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

Authors' Addresses

Michael H. Behringer (editor)
Cisco Systems
Building D, 45 Allee des Ormes
Mougins 06250
France

Email: mbehring@cisco.com

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

Email: tte+ietf@cs.fau.de

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

Email: sbjarnason@arbor.net