

ANIMA
Internet-Draft
Intended status: Informational
Expires: January 4, 2018

M. Behringer, Ed.

B. Carpenter
Univ. of Auckland
T. Eckert
Futurewei Technologies Inc.
L. Ciavaglia
P. Peloso
Nokia
B. Liu
Huawei Technologies
J. Nobre
Federal University of Rio Grande do Sul
J. Strassner
Huawei Technologies
July 3, 2017

A Reference Model for Autonomic Networking
draft-ietf-anima-reference-model-04

Abstract

This document describes a reference model for Autonomic Networking. The goal is to define how the various elements in an autonomic context work together, to describe their interfaces and relations. While the document is written as generally as possible, the initial solutions are limited to the chartered scope of the WG.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

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

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents 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	3
2.	The Network View	4
3.	The Autonomic Network Element	5
3.1.	Architecture	5
3.2.	The Adjacency Table	6
3.3.	State Machine	8
3.3.1.	State 1: Factory Default	8
3.3.2.	State 2: Enrolled	8
3.3.3.	State 3: In ACP	9
4.	The Autonomic Networking Infrastructure	9
4.1.	Naming	9
4.2.	Addressing	10
4.3.	Discovery	11
4.4.	Signaling Between Autonomic Nodes	12
4.5.	Routing	12
4.6.	The Autonomic Control Plane	12
4.7.	Information Distribution (*)	13
5.	Security and Trust Infrastructure	13
5.1.	Public Key Infrastructure	13
5.2.	Domain Certificate	14
5.3.	The MASA	14
5.4.	Sub-Domains (*)	14
5.5.	Cross-Domain Functionality (*)	14
6.	Autonomic Service Agents (ASA)	14
6.1.	General Description of an ASA	14
6.2.	ASA Life-Cycle Management	16
6.3.	Specific ASAs for the Autonomic Network Infrastructure .	17
6.3.1.	The enrollment ASAs	17
6.3.2.	The ACP ASA	17
6.3.3.	The Information Distribution ASA (*)	18
7.	Management and Programmability	18

7.1.	How an AN Network Is Managed	18
7.2.	Intent (*)	19
7.3.	Aggregated Reporting (*)	19
7.4.	Feedback Loops to NOC(*)	20
7.5.	Control Loops (*)	20
7.6.	APIs (*)	21
7.7.	Data Model (*)	21
8.	Coordination Between Autonomic Functions (*)	22
8.1.	The Coordination Problem (*)	22
8.2.	A Coordination Functional Block (*)	23
9.	Security Considerations	24
9.1.	Consequences of a Distributed System	24
9.2.	Security Mechanisms	25
10.	IANA Considerations	25
11.	Acknowledgements	25
12.	References	25
	Authors' Addresses	26

[1.](#) Introduction

The document "Autonomic Networking - Definitions and Design Goals" [[RFC7575](#)] explains the fundamental concepts behind Autonomic Networking, and defines the relevant terms in this space, as well as a high level reference model. [[RFC7576](#)] provides a gap analysis between traditional and autonomic approaches.

This document defines this reference model with more detail, to allow for functional and protocol specifications to be developed in an architecturally consistent, non-overlapping manner. While the document is written as generally as possible, the initial solutions are limited to the chartered scope of the WG.

As discussed in [[RFC7575](#)], the goal of this work is not to focus exclusively on fully autonomic nodes or networks. In reality, most networks will run with some autonomic functions, while the rest of the network is traditionally managed. This reference model allows for this hybrid approach.

This document describes phase 1 of an Autonomic Networking solution, and covers primarily the WG items of the ANIMA WG as of July 2017. Sections marked with (*) do not represent chartered items at this time. New WG items will require an update to this document, or potentially a new document.

2. The Network View

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

Figure 1 shows the high level view of an Autonomic Network. It consists of a number of autonomic nodes, which interact directly with each other. Those autonomic nodes provide a common set of capabilities across the network, called the "Autonomic Networking Infrastructure" (ANI). The ANI provides functions like naming, addressing, negotiation, synchronization, discovery and messaging.

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

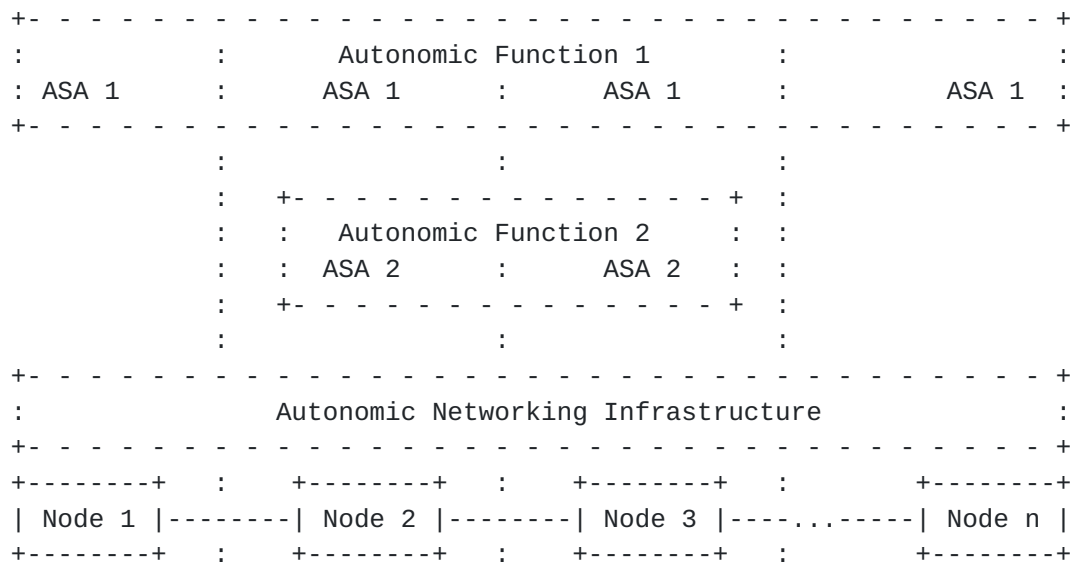


Figure 1: High level view of an Autonomic Network

In a horizontal view, autonomic functions span across the network, as well as the Autonomic Networking Infrastructure. In a vertical view, a node always implements the ANI, plus it may have one or several Autonomic Service Agents.

The Autonomic Networking Infrastructure (ANI) therefore is the foundation for autonomic functions. The current charter of the ANIMA WG is to specify the ANI, using a few autonomic functions as use cases.

3. The Autonomic Network Element

3.1. Architecture

This section describes an autonomic network element and its internal architecture. The reference model explained in the document "Autonomic Networking - Definitions and Design Goals" [[RFC7575](#)] shows the sources of information that an autonomic service agent can leverage: Self-knowledge, network knowledge (through discovery), Intent, and feedback loops. There are two levels inside an autonomic node: the level of Autonomic Service Agents, and the level of the Autonomic Networking Infrastructure, with the former using the services of the latter. Figure 2 illustrates this concept.

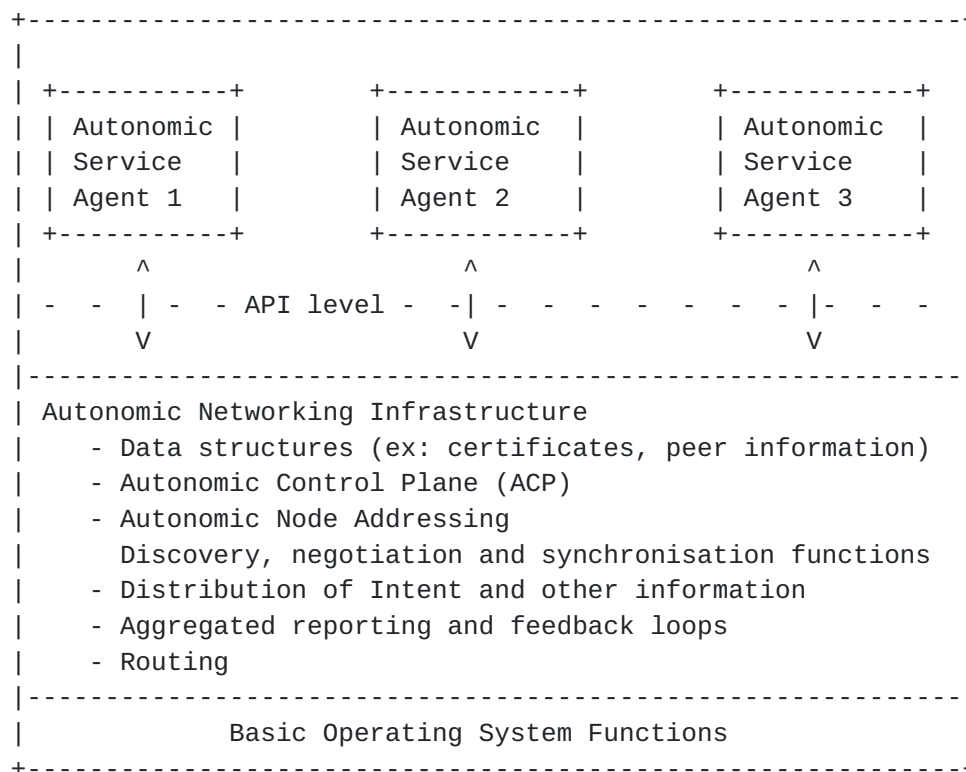


Figure 2: Model of an autonomic node

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

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

The "Basic Operating System Functions" include the "normal OS", including the network stack, security functions, etc.

Full AN nodes have the full Autonomic Networking Infrastructure, with the full functionality described in this document. At a later stage ANIMA may define a scope for constrained nodes with a reduced ANI and well-defined minimal functionality. They are currently out of scope.

3.2. The Adjacency Table

Autonomic Networking is based on direct interactions between devices of a domain. The Autonomic Networking Infrastructure (ANI) is normally built on a hop-by-hop basis. Therefore, many interactions in the ANI are based on the ANI adjacency table. There are interactions that provide input into the adjacency table, and other interactions that leverage the information contained in it.

The ANI adjacency table contains information about adjacent autonomic nodes, at a minimum: node-ID, IP address in data plane, IP address in ACP, domain, certificate. An autonomic node maintains this adjacency table up to date. The adjacency table only contains information about other nodes that are capable of Autonomic Networking; non-autonomic nodes are normally not tracked here. However, the information is tracked independently of the status of the peer nodes; specifically, it contains information about non-enrolled nodes, nodes of the same and other domains. The adjacency table may contain information about the validity and trust of the adjacent autonomic node's certificate, although all autonomic interactions must verify validity and trust independently.

The adjacency table is fed by the following inputs:

- 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 learn about all autonomic nodes around itself. This is described in [\[I-D.ietf-anima-grasp\]](#).
- o Vendor re-direct: A new device may receive information on where its home network is through a vendor based MASA re-direct; this is typically a routable address. See [\[I-D.ietf-anima-bootstrapping-keyinfra\]](#).

- 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 mechanisms 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 vendor MASA re-direct, which will be entered into the adjacency table (see second bullet above). See [\[I-D.ietf-anima-bootstrapping-keyinfra\]](#).
- 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 discovery [\[I-D.ietf-anima-grasp\]](#) to find Registrar(s) of its domain and potentiall other services.
- o If the node is part of an ACP and has discovered via GRASP at least one Registrar in its domain, it will start the "join assistant" ASA, and act as a join assistant for neighboring nodes that need to be bootstrapped. See [\[I-D.ietf-anima-bootstrapping-keyinfra\]](#).
- 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.

3.3.1. State 1: Factory Default

An autonomic node is leaving the factory in this state. In this state, the node has no domain specific configuration, specifically no LDevID, and could be used in any particular target network. It does however have a vendor/manufacture 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 enrolls into a domain; as part of this process it receives a domain identity (LDevID). If enrollment is successful, the next state is state 2. See [[I-D.ietf-anima-bootstrapping-keyinfra](#)] [Section 3](#) for details on enrollment.

3.3.2. State 2: Enrolled

An autonomic node is in the state "enrolled" if it has a domain identity (LDevID). It may have further configuration or state, for example if it had been in state 3 before, but lost all its ACP channels. The LDevID can only be removed from a device through a factory reset, which also removes all other state from the device. This ensures that a device has no stale domain specific state when entering the "enrolled" state from state 1.

Transitions:

- 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.

3.3.3. State 3: In ACP

In this state, the autonomic node has at least one ACP channel to another device. It can participate in further autonomic transactions, such as starting autonomic service agents. For example it must now enable the join assistant ASA, to help other devices to join the domain. Other conditions may apply to such interactions, for example to serve as a join assistant, the device must first discover a bootstrap Registrar.

Transitions:

- 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.

4. The Autonomic Networking Infrastructure

The Autonomic Networking Infrastructure provides a layer of common functionality across an Autonomic Network. It comprises "must implement" functions and services, as well as extensions.

An Autonomic Function, comprising of Autonomic Service Agents on nodes, can rely on the fact that all nodes in the network implement at least the "must implement" functions.

4.1. Naming

Inside a domain, each autonomic device should be assigned a unique name. The naming scheme should be consistent within a domain. Names are typically assigned by a Registrar at bootstrap time and persistent over the lifetime of the device. All Registrars in a domain must follow the same naming scheme.

In the absence of a domain specific naming scheme, a default naming scheme should use the same logic as the addressing scheme discussed in [[I-D.ietf-anima-autonomic-control-plane](#)]. The device name is then composed of a Registrar ID (for example taking a MAC address of the Registrar) and a device number. An example name would then look like this:

0123-4567-89ab-0001

The first three fields are the MAC address, the fourth field is the sequential number for the device.

4.2. Addressing

Autonomic Service Agents (ASAs) need to communicate with each other, using the autonomic addressing of the Autonomic Networking Infrastructure of the node they reside on. This section describes the addressing approach of the Autonomic Networking Infrastructure, used by ASAs.

Out of scope are addressing approaches for the data plane of the network, which may be configured and managed in the traditional way, or negotiated as a service of an ASA. One use case for such an autonomic function is described in [\[I-D.ietf-anima-prefix-management\]](#).

Autonomic addressing is a function of the Autonomic Networking Infrastructure (lower part of Figure 2), specifically the Autonomic Control Plane. ASAs do not have their own addresses. They may use either API calls, or the autonomic addressing scheme of the Autonomic Networking Infrastructure.

An autonomic addressing scheme has the following requirements:

- 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 Nodes may support Autonomic Service Agents in different virtual machines or containers. The addressing scheme should 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 with autonomic nodes outside their own domain. Therefore, an Autonomic Control Plane relies on a routing function. For Autonomic Networks to be interoperable, they must all support one common routing protocol.

The routing protocol is defined in the ACP document [[I-D.ietf-anima-autonomic-control-plane](#)].

4.6. The Autonomic Control Plane

The totality of autonomic interactions forms the "Autonomic Control Plane". This control plane can be either implemented in the global routing table of a node, such as IGP in today's networks; or it can be provided as an overlay network. The document "An Autonomic Control Plane" ([[I-D.ietf-anima-autonomic-control-plane](#)]) describes the details.

4.7. Information Distribution (*)

Certain forms of information, such as Intent, must be distributed across an autonomic domain. The distribution of information is also a function of the Autonomic Control Plane. One form of such information is Intent. Intent is the policy language of an Autonomic Network; see [Section 7.2](#) for general information on Intent. It is a high level policy, and should change only infrequently (order of days). Therefore, information such as Intent should be simply flooded to all nodes in an autonomic domain, and there is currently no perceived need to have more targeted distribution methods. Intent is also expected to be monolithic, and flooded as a whole. One possible method for distributing Intent, as well as other forms of data, is discussed in [[I-D.liu-anima-grasp-distribution](#)]. Intent and information distribution are not part of phase 1 of ANIMA.

5. Security and Trust Infrastructure

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

Autonomic nodes have direct interactions between themselves, which must be secured. Since an autonomic network does not rely on configuration, it is not an option to configure for example pre-shared keys. A trust infrastructure such as a PKI infrastructure must be in place. This section describes the principles of this trust infrastructure.

The default method to automatically bring up a trust infrastructure is defined in the document "Bootstrapping Key Infrastructures" [[I-D.ietf-anima-bootstrapping-keyinfra](#)]. The ASAs required for this enrollment process are described in [Section 6.3](#). An autonomic node must implement the enrollment and join assistant ASAs. The registrar ASA may be implemented only on a sub-set of nodes.

5.1. Public Key Infrastructure

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

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

5.2. Domain Certificate

Each device in an autonomic domain uses a domain certificate to prove its identity. [[I-D.ietf-anima-bootstrapping-keyinfra](#)] describes how a new device receives a domain certificate, and the certificate format.

5.3. The MASA

The Manufacturer Authorized Signing Authority (MASA) is a trusted service for bootstrapping devices. The purpose of the MASA is to provide ownership tracking of devices in a domain. The MASA provides audit, authorization, and ownership tokens to the registrar during the bootstrap process to assist in the authentication of devices attempting to join an Autonomic Domain, and to allow a joining device to validate whether it is joining the correct domain. The details for MASA service, security, and usage are defined in [[I-D.ietf-anima-bootstrapping-keyinfra](#)].

5.4. Sub-Domains (*)

By default, sub-domains are treated as different domains. This implies no trust between a domain and its sub-domains, and no trust between sub-domains of the same domain. Specifically, no ACP is built, and Intent is valid only for the domain it is defined for explicitly.

In phase 2 of ANIMA, alternative trust models should be defined, for example to allow full or limited trust between domain and sub-domain.

5.5. Cross-Domain Functionality (*)

By default, different domains do not interoperate, no ACP is built and no trust is implied between them.

In the future, models can be established where other domains can be trusted in full or for limited operations between the domains.

6. Autonomic Service Agents (ASA)

This section describes how autonomic services run on top of the Autonomic Networking Infrastructure.

6.1. General Description of an ASA

An Autonomic Service Agent (ASA) is defined in [[RFC7575](#)] as "An agent implemented on an autonomic node that implements an autonomic function, either in part (in the case of a distributed function) or

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

Thus we can distinguish at least three classes of ASAs:

- o Simple ASAs with a small footprint that could run anywhere.
- o Complex, 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, will be self-aware. Every autonomic node will be loaded with various functions and ASAs and will be aware of its own capabilities, typically decided by the hardware, firmware or pre-installed software. Its exact role may depend on Intent and on the surrounding network behaviors, which may include forwarding behaviors, aggregation properties, topology location, bandwidth, tunnel or translation properties, etc. The surrounding topology will depend on the network planning. Following an initial discovery phase, the device properties and those of its neighbors are the foundation of the behavior of a specific device. A device and its ASAs have no pre-configuration for the particular network in which they are installed.

Since all ASAs will interact with the ANI, they will depend on appropriate application programming interfaces (APIs). It is desirable that ASAs are portable between operating systems, so these APIs need to be universal. An API for GRASP is described in [[I-D.liu-anima-grasp-api](#)].

ASAs will in general be designed and coded by experts in a particular technology and use case, not by experts in the ANI and its components. Also, they may be coded in a variety of programming languages, in particular including languages that support object constructs as well as traditional variables and structures. The APIs should be designed with these factors in mind.

It must be possible to run ASAs as non-privileged (user space) processes except for those (such as the infrastructure ASAs) that necessarily require kernel privilege. Also, it is highly desirable that ASAs can be dynamically loaded on a running node.

Since autonomic systems must be self-repairing, it is of great importance that ASAs are coded using robust programming techniques. All run-time error conditions must be caught, leading to suitable recovery actions, with a complete restart of the ASA as a last resort. Conditions such as discovery failures or negotiation failures must be treated as routine, with the ASA retrying the failed operation, preferably with an exponential back-off in the case of persistent errors. When multiple threads are started within an ASA, these threads must be monitored for failures and hangups, and appropriate action taken. Attention must be given to garbage collection, so that ASAs never run out of resources. There is assumed to be no human operator - again, in the worst case, every ASA must be capable of restarting itself.

ASAs will automatically benefit from the security provided by the ANI, and specifically by the ACP and by GRASP. However, beyond that, they are responsible for their own security, especially when communicating with the specific targets of their function. Therefore, the design of an ASA must include a security analysis beyond 'use ANI security.'

6.2. ASA Life-Cycle Management

ASAs operating on a given ANI may come from different providers and pursue different objectives. Whichever the ASA, its management and its interactions with the ANI must follow the same operating principles, hence comply to a generic life-cycle management model.

The ASA life-cycle provides standard processes to:

- o install ASA: copy the ASA code onto the host 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-cycle 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 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 assistant ASA (see below). Such a node is known as a Pledge during the enrollment process. This ASA must be installed by default on all nodes that require an autonomic zero-touch bootstrap.

6.3.1.2. The Join Assistant ASA

This ASA includes the function of an autonomic node that helps a non-enrolled, adjacent devices to enroll into the domain. This ASA must be installed on all nodes, although only one join assistant needs to be active on a given LAN.

6.3.1.3. The Join Registrar ASA

This ASA includes the join registrar function in an autonomic network. This ASA does not need to be installed on all nodes, but only on nodes that implement the Join Registrar function.

6.3.2. The ACP ASA

This ASA includes the ACP function in an autonomic network. In particular it acts to discover other potential ACP nodes, and to support the establishment and teardown of ACP channels. This ASA

must be installed on all nodes. For details see [Section 4.6](#) and [\[I-D.ietf-anima-autonomic-control-plane\]](#).

6.3.3. The Information Distribution ASA (*)

This ASA is currently out of scope in ANIMA, and provided here only as background information.

This ASA includes the information distribution function in an autonomic network. In particular it acts to announce the availability of Intent and other information to all other autonomic nodes. This ASA does not need to be installed on all nodes, but only on nodes that implement the information distribution function. For details see [Section 4.7](#).

7. Management and Programmability

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

7.1. How an AN Network Is Managed

Autonomic management usually co-exists with traditional management methods in most networks. Thus, autonomic behavior will be defined for individual functions in most environments. In fact, the co-existence is twofold: autonomic functions can use traditional methods and protocols (e.g., SNMP and NETCONF) to perform management tasks; and autonomic functions can conflict with behavior enforced by the same traditional methods and protocols.

The autonomic Intent is defined at a high level of abstraction. However, since it is necessary to address individual managed elements, autonomic management needs to communicate in lower-level interactions (e.g., commands and requests). For example, it is expected that the configuration of such elements be performed using NETCONF and YANG modules as well as the monitoring be executed through SNMP and MIBs.

Conflict can occur between autonomic default behavior, autonomic Intent, traditional management methods. Conflict resolution is achieved in autonomic management through prioritization [\[RFC7575\]](#). The rationale is that manual and node-based management have a higher priority over autonomic management. Thus, the autonomic default behavior has the lowest priority, then comes the autonomic Intent (medium priority), and, finally, the highest priority is taken by node-specific network management methods, such as the use of command line interfaces.

7.2. Intent (*)

Intent is not covered by the ANIMA charter at the time of this writing. This section is for informational purposes only.

This section gives an overview of Intent, and how it is managed. Intent and Policy-Based Network Management (PBNM) is already described inside the IETF (e.g., PCIM and SUPA) and in other SDOs (e.g., DMTF and TMF ZOOM).

Intent can be described as an abstract, declarative, high-level policy used to operate an autonomic domain, such as an enterprise network [[RFC7575](#)]. Intent should be limited to high level guidance only, thus it does not directly define a policy for every network element separately. It is expected Intent definitions from autonomic function(s) and even from traditional network management elements.

Intent can be refined to lower level policies using different approaches. This is expected in order to adapt the Intent to the capabilities of managed devices. Intent may contain role or function information, which can be translated to specific nodes [[RFC7575](#)]. One of the possible refinements of the Intent is using Event-Condition-Action (ECA) rules.

Different parameters may be configured for Intent. These parameters are usually provided by the human operator. Some of these parameters can influence the behavior of specific autonomic functions as well as the way the Intent is used to manage the autonomic domain.

Intent is discussed in more detail in [[I-D.du-anima-an-intent](#)]. Intent as well as other types of information are distributed via GRASP, see [[I-D.liu-anima-grasp-distribution](#)].

7.3. Aggregated Reporting (*)

At the time of this writing, aggregated reporting is not in the ANIMA charter. This section is provided for information only.

Autonomic Network should minimize the need for human intervention. In terms of how the network should behave, this is done through an autonomic Intent provided by the human administrator. In an analogous manner, the reports which describe the operational status of the network should aggregate the information produced in different network elements in order to present the effectiveness of autonomic Intent enforcement. Therefore, reporting in an autonomic network should happen on a network-wide basis [[RFC7575](#)].

Several events can occur in an autonomic network in the same way they can happen in a traditional network. However, when reporting to a human administrator, such events should be aggregated to avoid advertisement about individual managed elements. In this context, algorithms may be used to determine what should be reported (e.g., filtering) and in which way and how different events are related to each other. Besides that, an event in an individual element can be compensated by changes in other elements to maintain a network-wide level which is described in the autonomic Intent.

Reporting in an autonomic network may be in the same abstraction level of the Intent. In this context, the visibility on current operational status of an autonomic network can be used to switch to different management modes. Despite the fact that autonomic management should minimize the need for user intervention, possibly there are some events that need to be addressed by human administrator actions.

7.4. Feedback Loops to NOC(*)

Feedback loops are required in an autonomic network to allow the intervention of a human administrator or central control systems, while maintaining a default behaviour. Through a feedback loop an administrator can be prompted with a default action, and has the possibility to acknowledge or override the proposed default action.

7.5. Control Loops (*)

Control loops are used in autonomic networking to provide a generic mechanism to enable the Autonomic System to adapt (on its own) to various factors that can change the goals that the Autonomic System 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.

The document [[I-D.strassner-anima-control-loops](#)] defines the requirements for an autonomic control loop, describes different types of control loops, and explains how control loops are used in an autonomic system.

7.6. APIs (*)

Most APIs are static, meaning that they are pre-defined and represent an invariant mechanism for operating with data. An Autonomic Network SHOULD be able to use dynamic APIs in addition to static APIs.

A dynamic API is one that retrieves data using a generic mechanism, and then enables the client to navigate the retrieved data and operate on it. Such APIs typically use introspection and/or reflection. Introspection enables software to examine the type and properties of an object at runtime, while reflection enables a program to manipulate the attributes, methods, and/or metadata of an object.

APIs MUST be able to express and preserve semantics across different domains. 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 (*)

The following definitions are taken from [supa-model]:

An information model is a representation of concepts of interest to an environment in a form that is independent of data repository, data definition language, query language, implementation language, and protocol. In contrast, a data model is a representation of concepts of interest to an environment in a form that is dependent on data repository, data definition language, query language, implementation language, and protocol (typically, but not necessarily, all three).

The utility of an information model is to define objects and their relationships in a technology-neutral manner. This forms a consensual vocabulary that the ANI and ASAs can use. A data model is then a technology-specific mapping of all or part of the information model to be used by all or part of the system.

A system may have multiple data models. Operational Support Systems, for example, typically have multiple types of repositories, such as SQL and NoSQL, to take advantage of the different properties of each. If multiple data models are required by an Autonomic System, then an information model SHOULD be used to ensure that the concepts of each data model can be related to each other without technological bias.

A data model is essential for certain types of functions, such as a MRACL. More generally, a data model can be used to define the objects, attributes, methods, and relationships of a software system (e.g., the ANI, an autonomic node, or an ASA). A data model can be used to help design an API, as well as any language used to interface to the Autonomic Network.

8. Coordination Between Autonomic Functions (*)

8.1. The Coordination Problem (*)

Different autonomic functions may conflict in setting certain parameters. For example, an energy efficiency function may want to shut down a redundant link, while a load balancing function would not want that to happen. The administrator must be able to understand and resolve such interactions, to steer autonomic network performance to a given (intended) operational point.

Several interaction types may exist among autonomic functions, for example:

- 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 exists 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

9.1. Consequences of a Distributed System

An autonomic network consists of autonomic devices that form a distributed self-managing system. Devices within a domain share a common trust anchor and thus implicitly trust each other. This means that any device inside a trust domain can by default use all distributed functions in the entire autonomic domain in a malicious way.

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 is covered in detail in [[I-D.ietf-anima-bootstrapping-keyinfra](#)].
- 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. However, the distributed nature of AN, specifically the Autonomic Control Plane, increases the threat surface. 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.

Most AN messages run inside the encrypted ACP. The not protected AN messages outside the ACP are limited to simple discovery methods. See Section 2.5.2 of [[I-D.ietf-anima-grasp](#)] for a list of such messages. If AN messages can be observed by a third party, they might reveal valuable information about network configuration, security precautions in use, individual users, and their traffic

patterns. If encrypted, AN messages might still reveal some information via traffic analysis, but this would be quite limited (for example, this would be highly unlikely to reveal any specific information about user traffic).

9.2. Security Mechanisms

The components of the ANI must each include appropriate security mechanisms. In particular, the ACP must provide security against interception, forgery, and replay of any messages sent over the ACP. The signaling protocol may rely on this protection, but must also provide for security when running without an ACP. All components of the security bootstrap process must of course themselves be secured. All ASAs must make use of the ANI's security, and must be carefully designed so that they do not create security "holes" in the boundary of the whole AN system.

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.

12. References

[I-D.du-anima-an-intent]

Du, Z., Jiang, S., Nobre, J., Ciavaglia, L., and M. Behringer, "ANIMA Intent Policy and Format", [draft-du-anima-an-intent-05](#) (work in progress), February 2017.

[I-D.ietf-anima-autonomic-control-plane]

Behringer, M., Eckert, T., and S. Bjarnason, "An Autonomic Control Plane", [draft-ietf-anima-autonomic-control-plane-06](#) (work in progress), March 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-06](#) (work in progress), May 2017.

[I-D.ietf-anima-grasp]

Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", [draft-ietf-anima-grasp-14](#) (work in progress), July 2017.

[I-D.ietf-anima-prefix-management]

Jiang, S., Du, Z., Carpenter, B., and Q. Sun, "Autonomic IPv6 Edge Prefix Management in Large-scale Networks", [draft-ietf-anima-prefix-management-04](#) (work in progress), June 2017.

[I-D.liu-anima-grasp-api]

Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)", [draft-liu-anima-grasp-api-04](#) (work in progress), June 2017.

[I-D.liu-anima-grasp-distribution]

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

[I-D.strassner-anima-control-loops]

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

[IDevID] IEEE Standard, , "IEEE 802.1AR Secure Device Identifier", December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.

[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>>.

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
Villardeaux
Nozay 91460
FR

Email: laurent.ciavaglia@nokia.com

Peloso Pierre
Nokia
Villardeaux
Nozay 91460
FR

Email: pierre.peloso@nokia.com

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

Email: leo.liubing@huawei.com

Jeferson Campos Nobre
Federal University of Rio Grande do Sul
Av. Bento Goncalves, 9500
Porto Alegre 91501-970
Brazil

Email: jcnobre@inf.ufrgs.br

John Strassner
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: john.sc.strassner@huawei.com

