ï»¿

Internet Engineering Task Force (IETF)
Request for Comments: 8993
Category: Informational
ISSN: 2070-1721

M. Behringer, Ed.

B. Carpenter
Univ. of Auckland
T. Eckert
Futurewei USA
L. Ciavaglia
Nokia
J. Nobre
UFRGS
May 2021

A Reference Model for Autonomic Networking

Abstract

This document describes a reference model for Autonomic Networking for managed networks. It defines the behavior of an autonomic node, how the various elements in an autonomic context work together, and how autonomic services can use the infrastructure.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

This document defines this reference model with more detail to allow for functional and protocol specifications to be developed in an architecturally consistent, non-overlapping manner.

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

For example, it is possible in an existing, non-autonomic network to enroll devices in a traditional way to bring up a trust infrastructure with certificates. This trust infrastructure could then be used to automatically bring up an Autonomic Control Plane (ACP) and run traditional network operations over the secure and self-healing ACP. See [RFC8368] for a description of this use case.

The scope of this model is therefore limited to networks that are to some extent managed by skilled human operators, loosely referred to as "professionally managed" networks. Unmanaged networks raise additional security and trust issues that this model does not cover.

This document describes the first phase of an Autonomic Networking solution that is both simple and implementable. It is expected that the experience from this phase will be used in defining updated and extended specifications over time. Some topics are considered architecturally in this document but are not yet reflected in the implementation specifications. They are marked with an (*).

2. Network View

This section describes the various elements in a network with autonomic functions and explains 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 (ASAs)", which are instantiated on nodes.

```
   +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
   :            :       Autonomic Function 1        :                 :
   : ASA 1      :      ASA 1      :      ASA 1      :          ASA 1  :
   +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
                :                 :                 :
                :   +- - - - - - - - - - - - - - +  :
                :   :   Autonomic Function 2     :  :
                :   :  ASA 2      :      ASA 2   :  :
                :   +- - - - - - - - - - - - - - +  :
                :                 :                 :
   +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
   :               Autonomic Networking Infrastructure                :
   +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
   +--------+   :    +--------+   :    +--------+   :        +--------+
   | Node 1 |--------| Node 2 |--------| Node 3 |----...-----| Node n |
   +--------+   :    +--------+   :    +--------+   :        +--------+
```

Figure 1: High-Level View of an Autonomic Network

In a horizontal view, autonomic functions span across the network, as well as the ANI. In a vertical view, a node always implements the ANI, plus it may have one or several ASAs. ASAs may be standalone or use other ASAs in a hierarchical way.

Therefore, the ANI is the foundation for autonomic functions.

3. Autonomic Network Element

This section explains the general architecture of an Autonomic Network element (Section 3.1), how it tracks its surrounding environment in an adjacency table (Section 3.2), and the state machine that defines the behavior of the network element (Section 3.3), based on that adjacency table.

3.1. Architecture

This section describes an Autonomic Network element and its internal architecture. The reference model explained in the document "Autonomic Networking: Definitions and Design Goals" [RFC7575] shows the sources of information that an ASA can leverage: self-knowledge, network knowledge (through discovery), Intent (see Section 7.2), and feedback loops. There are two levels inside an autonomic node: the level of ASAs and the level of the ANI, with the former using the services of the latter. Figure 2 illustrates this concept.

```
   +------------------------------------------------------------+
   |                                                            |
   | +-----------+        +------------+        +------------+  |
   | | Autonomic |        | Autonomic  |        | Autonomic  |  |
   | | Service   |        | Service    |        | Service    |  |
   | | Agent 1   |        | Agent 2    |        | Agent 3    |  |
   | +-----------+        +------------+        +------------+  |
   |       ^                    ^                     ^         |
   | -  -  | -  - API level -  -| -  -  -  -  -  -  - |-  -  -  |
   |       V                    V                     V         |
   |------------------------------------------------------------|
   | Autonomic Networking Infrastructure                        |
   |    - Data structures (ex: certificates, peer information)  |
   |    - Generalized Autonomic Control Plane (GACP)            |
   |    - Autonomic node addressing and naming                  |
   |    - Discovery, negotiation and synchronization functions  |
   |    - Distribution of Intent and other information          |
   |    - Aggregated reporting and feedback loops               |
   |    - Routing                                               |
   |------------------------------------------------------------|
   |             Basic Operating System Functions               |
   +------------------------------------------------------------+
```

Figure 2: Model of an Autonomic Node

The ANI (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 ASAs (upper part of Figure 2). It contains addressing and naming of autonomic nodes, discovery, negotiation and synchronization functions, distribution of information, reporting, feedback loops, and routing inside the ACP.

The Generalized ACP (GACP) is the summary of all interactions of the ANI with other nodes and services. A specific implementation of the GACP is referred to here as the ACP and described in [RFC8994].

The use cases of "Autonomics" (such as self-management, self-optimization, etc.) are implemented as ASAs. They use the services and data structures of the underlying ANI, which should be self-managing.

The Basic Operating System Functions (lower part of Figure 2) include the normal OS (e.g., the network stack and security functions).

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

3.2. Adjacency Table

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

The ANI adjacency table contains, at a minimum, information about adjacent autonomic nodes: Node-ID, IP address in data plane, IP address in ACP, domain, and 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, the adjacency table contains information about non-enrolled nodes of the same and other domains. The adjacency table may contain information about the validity and trust level of the adjacent autonomic nodes.

The adjacency table is fed by the following inputs:

* 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 node learns about all autonomic nodes around itself. The related Standards Track documents ([RFC8990], [RFC8995], and [RFC8994]) describe in detail how link-local discovery is used.

* Vendor redirect: A new device may receive information on where its home network is through a vendor-based Manufacturer Authorized Signing Authority (MASA) (see Section 5.3) redirect; this is typically a routable address.

* 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 bypass a non-autonomic device or network. As this pertains to new devices, it is covered in Appendices A and B of [RFC8995].

The adjacency table defines the behavior of an autonomic node:

* If the node has not bootstrapped into a domain (i.e., doesn't have a domain certificate), it rotates through all nodes in the adjacency table that claim to have a domain and will attempt bootstrapping through them, one by one. One possible response is a redirect via a vendor MASA, which will be entered into the adjacency table (see second bullet above). See [RFC8995] for details.

* If the adjacent node has the same domain, it will authenticate that adjacent node and, if successful, establish the ACP. See [RFC8994].

* Once the node is part of the ACP of a domain, it will use GRASP [RFC8990] to find the registrar(s) of its domain and potentially other services.

* If the node is part of an ACP and has discovered at least one registrar in its domain via GRASP, it will start the join proxy ASA and act as a join proxy for neighboring nodes that need to be bootstrapped. See Section 6.3.1.2 for details.

* Other behaviors are possible, for example, establishing the ACP with devices of a subdomain or other domains. These will likely be controlled by Intent and are outside the scope of this document. Note that Intent is distributed through the ACP; therefore, a node can only adapt Intent-driven behavior once it has joined the ACP. At the moment, the ANIMA Working Group 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 and synchronization via GRASP [RFC8990] are 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 ASAs. They run logically above the ANI and may use the adjacency table, the ACP, negotiation and synchronization through GRASP in the ACP, Intent, and other functions of the ANI. Since the ANI only provides autonomic interactions within a domain, autonomic functions can also use any other context on a node, specifically the global data plane.

3.3. State Machine

Autonomic Networking applies during the full life cycle of a node. This section describes a state machine of an autonomic node throughout its life.

A device is normally expected to store its domain-specific identity, the Local Device Identifier (LDevID) (see Section 5.2), in persistent storage to be available after a power-cycle event. For device types that cannot store the LDevID in persistent storage, a power-cycle event is effectively equivalent to a factory reset.

3.3.1. State 1: Factory Default

An autonomic node leaves the factory in this state. In this state, the node has no domain-specific configuration, specifically no LDevID, and could be used in any particular target network. It does, however, have a vendor/manufacturer-specific ID, the Initial Device Identifier (IDevID) [IDevID]. Nodes without IDevID cannot be autonomically and securely enrolled into a domain; they require manual pre-staging, in which case the pre-staging takes them directly to state 2.

Transitions:

* Bootstrap event: The device 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 [RFC8995] for details on enrollment.

* Power-cycle event: The device loses all state tables. It remains in state 1.

3.3.2. State 2: Enrolled

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

Transitions:

* Joining ACP: The device establishes an ACP channel to an adjacent device. See [RFC8994] for details. Next state: 3.

* Factory reset: A factory reset removes all configuration and the domain identity (LDevID) from the device. Next state: 1.

* Power-cycle event: The device loses all state tables, but not its domain identity (LDevID). It remains in state 2.

3.3.3. State 3: In ACP

In this state, the autonomic node has at least one ACP channel to another device. The node can now participate in further autonomic transactions, such as starting ASAs (e.g., it must now enable the join proxy ASA, to help other devices to join the domain). Other conditions may apply to such interactions, for example, to serve as a join proxy, the device must first discover a bootstrap registrar.

Transitions:

* Leaving ACP: The device drops the last (or only) ACP channel to an adjacent device. Next state: 2.

* Factory reset: A factory reset removes all configuration and the domain identity (LDevID) from the device. Next state: 1.

* Power-cycle event: The device loses all state tables but not its domain identity (LDevID). Next state: 2.

4. Autonomic Networking Infrastructure

The ANI provides a layer of common functionality across an Autonomic Network. It provides the elementary functions and services, as well as extensions. An autonomic function, comprising of ASAs on nodes, uses the functions described in this section.

4.1. Naming

Inside a domain, each autonomic device should be assigned a unique name. The naming scheme should be consistent within a domain. Names are typically assigned by a registrar at bootstrap time and are 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 [RFC8994]. The device name is then composed of a Registrar-ID (for example, taking a Media Access Control (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, and the fourth field is the sequential number for the device.

4.2. Addressing

ASAs need to communicate with each other, using the autonomic addressing of the ANI of the node they reside on. This section describes the addressing approach of the ANI used by ASAs.

Addressing approaches for the data plane of the network are outside the scope of this document. These addressing approaches may be configured and managed in the traditional way or negotiated as a service of an ASA. One use case for such an autonomic function is described in [RFC8992].

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

An autonomic addressing scheme has the following requirements:

* Zero-touch for simple networks: Simple networks should have complete self-management of addressing and not require any central address management, tools, or address planning.

* Low-touch for complex networks: If complex networks require operator input for autonomic address management, it should be limited to high-level guidance only, expressed in Intent.

* Flexibility: The addressing scheme must be flexible enough for nodes to be able to move around and for the network to grow, split, and merge.

* Robustness: It should be as hard as possible for an administrator to negatively affect addressing (and thus connectivity) in the autonomic context.

* Stability: The addressing scheme should be as stable as possible. However, implementations need to be able to recover from unexpected address changes.

* Support for virtualization: Autonomic functions can exist either

at the level of the physical network and physical devices or at the level of virtual machines, containers, and networks. In particular, autonomic nodes may support ASAs in virtual entities. The infrastructure, including the addressing scheme, should be able to support this architecture.

* Simplicity: The addressing scheme should be simple to make engineering easier and to give the human administrator an easy way to troubleshoot autonomic functions.

* Scale: The proposed scheme should work in any network of any size.

* Upgradability: The scheme must be able to support different addressing concepts in the future.

The proposed addressing scheme is described in the document "An Autonomic Control Plane (ACP)" [RFC8994].

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; therefore, it needs to discover peers and other resources in the network. This section describes various discovery functions in an Autonomic Network.

First, discovering nodes and their properties and capabilities is 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.

Second, network services such as Authentication, Authorization, and Accounting (AAA) should also be discovered and not configured. Service discovery is required for such tasks. An Autonomic Network can leverage existing service discovery functions, use a new approach, or use a mixture.

Thus, the discovery mechanism could either be fully integrated with autonomic signaling (next section) or use an independent discovery mechanism such as DNS-based Service Discovery or the Service Location Protocol. This choice could be made independently for each ASA, 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 [RFC8990] for discovery.

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 "GeneRic Autonomic Signaling Protocol (GRASP)" [RFC8990] describes more detailed requirements for discovery, negotiation, and synchronization in an Autonomic Network. It also defines a protocol, called GRASP, for this purpose; GRASP includes an integrated but optional discovery process.

GRASP is normally expected to run inside the 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 in later phases, they must also be able to communicate with autonomic nodes outside their own domain. Therefore, an ACP 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 [RFC8994].

4.6. Autonomic Control Plane

The ACP carries the control protocols in an Autonomic Network. In the architecture described in this document, it is implemented as an overlay network. The document "An Autonomic Control Plane (ACP)" [RFC8994] describes the implementation details suggested in this document. This document uses the term "overlay" to mean a set of point-to-point adjacencies congruent with the underlying interconnection topology. The terminology may not be aligned with a common usage of the term "overlay" in the routing context. See [RFC8368] for uses cases for the ACP.

4.7. Information Distribution (*)

Certain forms of information require distribution across an autonomic domain. The distribution of information runs inside the ACP. For example, Intent is distributed across an autonomic domain, as explained in [RFC7575].

Intent is the policy language of an Autonomic Network (see also Section 7.2). It is a high-level policy and should change only infrequently (order of days). Therefore, information such as Intent should be simply flooded to all nodes in an autonomic domain, and there is currently no perceived need to have more targeted distribution methods. Intent is also expected to be monolithic and flooded as a whole. One possible method for distributing Intent, as well as other forms of data, is discussed in [GRASP-DISTRIB]. Intent and information distribution are not part of the ANIMA Working Group charter.

5. Security and Trust Infrastructure

An Autonomic Network is self-protecting. All protocols are secure by default, without the requirement for the administrator to explicitly configure security, with the exception of setting up a PKI infrastructure.

Autonomic nodes have direct interactions between themselves, which must be secured. Since an Autonomic Network does not rely on configuration, it is not an option to configure, for example, pre-shared keys. A trust infrastructure such as a PKI infrastructure must be in place. This section describes the principles of this trust infrastructure. In this first phase of Autonomic Networking, a device is either 1) within the trust domain and fully trusted or 2) outside the trust domain and fully untrusted.

The default method to automatically bring up a trust infrastructure is defined in the document "Bootstrapping Remote Secure Key Infrastructure (BRSKI)" [RFC8995]. The ASAs required for this enrollment process are described in Section 6.3. An autonomic node must implement the enrollment and join proxy ASAs. The registrar ASA may be implemented only on a subset of nodes.

5.1. Public Key Infrastructure

An autonomic domain uses a PKI model. The root of trust is a Certification Authority (CA). A registrar acts as a Registration Authority (RA).

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

5.2. Domain Certificate

Each device in an autonomic domain uses a domain certificate (LDevID) to prove its identity. A new device uses its manufacturer-provided certificate (IDevID) during bootstrap to obtain a domain certificate. [RFC8995] describes how a new device receives a domain certificate and defines the certificate format.

5.3. 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 [RFC8995].

5.4. Subdomains (*)

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

In the ANIMA Working Group charter, alternative trust models should be defined, for example, to allow full or limited trust between domain and subdomain.

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 (ASAs)

This section describes how autonomic services run on top of the ANI.

6.1. General Description of an ASA

An 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 GRASP [RFC8990] or otherwise. Of course, it also interacts with the specific targets of its function, using any suitable mechanism. Unless its function is very simple, the ASA will need to handle overlapping asynchronous operations. It may therefore be a quite complex piece of software in its own right, forming part of the application layer above the ANI. ASA design guidelines are available in [ASA-GUIDELINES].

Thus, we can distinguish at least three classes of ASAs:

* Simple ASAs with a small footprint that could run anywhere.

* Complex, possibly multi-threaded ASAs that have a significant resource requirement and will only run on selected nodes.

* A few 'infrastructure ASAs' that use basic ANI features in support of the ANI itself, which must run in all autonomic nodes. These are outlined in the following sections.

Autonomic nodes, and therefore their ASAs, know their own capabilities and restrictions, derived from hardware, firmware, or pre-installed software; they are "self-aware".

The role of an autonomic node depends on Intent and on the surrounding network behaviors, which may include forwarding behaviors, aggregation properties, topology location, bandwidth, tunnel or translation properties, etc. For example, a node may decide to act as a backup node for a neighbor, if its capabilities allow it to do so.

Following an initial discovery phase, the node's properties and those of its neighbors are the foundation of the behavior of a specific node. A node and its ASAs have no pre-configuration for the particular network in which they are installed.

Since all ASAs will interact with the ANI, they will depend on appropriate application programming interfaces (APIs). It is desirable that ASAs are portable between operating systems, so these APIs need to be universal. An API for GRASP is described in [RFC8991].

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, 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 runtime error conditions must be caught, leading to suitable minimally disruptive recovery actions, but a complete restart of the ASA must also be considered. 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, specifically by the ACP and by GRASP. However, beyond that, they are responsible for their own security, especially when communicating with the specific targets of their function. Therefore, the design of an ASA must include a security analysis beyond 'use ANI security'.

6.2. ASA Life-Cycle Management

ASAs operating on a given ANI may come from different providers and pursue different objectives. Management of ASAs and their interactions with the ANI should follow the same operating principles and thus comply to a generic life-cycle management model.

The ASA life cycle provides standard processes to:

* install ASA: copy the ASA code onto the node and start it.

* deploy ASA: associate the ASA instance with a (some) managed network device(s) (or network function).

* control ASA execution: when and how an ASA executes its control loop.

This life cycle will also define which interactions ASAs have with the ANI in between the different states. The noticeable interactions are:

* Self-description of ASA instances at the end of deployment: Its format needs to define the information required for the management of ASAs by ANI entities.

* Control of the ASA control loop during the operation: Signaling has to carry formatted messages to control ASA execution (at least starting and stopping the control loop).

6.3. Specific ASAs for the Autonomic Networking 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 (pledge, join proxy, join registrar) support together the trust enrollment process described in Section 5. For details see [RFC8995].

6.3.1. 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 a join proxy 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 Proxy ASA

This ASA includes the function of an autonomic node that helps non-enrolled, adjacent devices to enroll into the domain. This ASA must be installed on all nodes, although only one join proxy needs to be active on a given LAN. See also [RFC8995].

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. 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 [RFC8994].

6.3.3. Information Distribution ASA (*)

This ASA is currently out of scope in the ANIMA Working Group charter and is provided here only as background information.

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

Note that information distribution can be implemented as a function in any ASA. See [GRASP-DISTRIB] for more details on how information is suggested to be distributed.

7. Management and Programmability

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

7.1. Managing a (Partially) Autonomic Network

Autonomic management usually coexists with traditional management methods in most networks. Thus, autonomic behavior will be defined for individual functions in most environments. Examples of overlap are:

* Autonomic functions can use traditional methods and protocols (e.g., SNMP and the Network Configuration Protocol (NETCONF)) to perform management tasks, inside and outside the ACP.

* Autonomic functions can conflict with behavior enforced by the same traditional methods and protocols.

* Traditional functions can use the ACP, for example, if reachability on the data plane is not (yet) established.

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

Conflict can occur between autonomic default behavior, autonomic Intent, and 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 than autonomic management. Thus, the autonomic default behavior has the lowest priority, then comes the autonomic Intent (medium priority), and, finally, the highest priority is taken by node-specific network management methods, such as the use of command-line interfaces.

7.2. Intent (*)

Intent is not covered in the current implementation specifications. This section discusses a topic for further research.

This section gives an overview of Intent and how it is managed. Intent and Policy-Based Network Management (PBNM) is already described inside the IETF (e.g., Policy Core Information Model (PCIM)) and in other Standards Development Organizations (SDOs) (e.g., the Distributed Management Task Force (DMTF)).

Intent can be described as an abstract, declarative, high-level policy used to operate an autonomic domain, such as an enterprise network [RFC7575]. Intent should be limited to high-level guidance only; thus, it does not directly define a policy for every network element separately.

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

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

Intent is discussed in more detail in [ANIMA-INTENT]. Intent as well as other types of information are distributed via GRASP; see [GRASP-DISTRIB].

7.3. Aggregated Reporting (*)

Aggregated reporting is not covered in the current implementation specifications. This section discusses a topic for further research.

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

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

Reporting in an Autonomic Network may be at the same abstraction level as Intent. In this context, the aggregated view of the 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, some events may need to be addressed by the actions of a human administrator.

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 behavior. Through a feedback loop, an administrator must be prompted with a default action and has the possibility to acknowledge or override the proposed default action.

Unidirectional notifications to the Network Operations Center (NOC) that do not propose any default action and do not allow an override as part of the transaction are considered like traditional notification services, such as syslog. They are expected to coexist with autonomic methods but are not covered in this document.

7.5. Control Loops (*)

Control loops are not covered in the current implementation specifications. This section discusses a topic for further research.

Control loops are used in Autonomic Networking to provide a generic mechanism to enable the autonomic system to adapt (on its own) to various factors that can change the goals that the Autonomic Network is trying to achieve or how those goals are achieved. For example, as user needs, business goals, and the ANI itself changes, self-adaptation enables the ANI to change the services and resources it makes available to adapt to these changes.

Control loops operate to continuously observe and collect data that enables the autonomic management system to understand changes to the behavior of the system being managed and then provide actions to move the state of the system being managed toward a common goal. Self-adaptive systems move decision making from static, pre-defined commands to dynamic processes computed at runtime.

Most autonomic systems use a closed control loop with feedback. Such control loops should be able to be dynamically changed at runtime to adapt to changing user needs, business goals, and changes in the ANI.

7.6. APIs (*)

[RFC8991] defines a conceptual outline for an API for the GeneRic Autonomic Signaling Protocol (GRASP). This conceptual API is designed for ASAs to communicate with other ASAs through GRASP. Full Standards Track API specifications are not covered in the current implementation specifications.

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

APIs must be able to express and preserve the semantics of data models. For example, software contracts [Meyer97] are based on the principle that a software-intensive system, such as an Autonomic Network, is a set of communicating components whose interaction is based on precisely defined specifications of the mutual obligations that interacting components must respect. This typically includes specifying:

* pre-conditions that must be satisfied before the method can start execution

* post-conditions that must be satisfied when the method has finished execution

* invariant attributes that must not change during the execution of the method

7.7. Data Model (*)

Data models are not covered in the current implementation specifications. This section discusses a topic for further research.

The following definitions of "data model" and "information model" are adapted from [SUPA-DATA].

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.

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

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

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

8. Coordination between Autonomic Functions (*)

Coordination between autonomic functions is not covered in the current implementation specifications. This section discusses a topic for further research.

8.1. Coordination Problem (*)

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

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

* Cooperation: An autonomic function can improve the behavior or performance of another autonomic function, such as a traffic forecasting function used by a traffic allocation function.

* Dependency: An autonomic function cannot work without another one being present or accessible in the Autonomic Network.

* Conflict: A metric value conflict is a conflict where one metric is influenced by parameters of different autonomic functions. A parameter value conflict is a conflict where one parameter is modified by different autonomic functions.

Solving the coordination problem beyond one-by-one cases can rapidly become intractable for large networks. Specifying a common functional block on coordination is a first step to address the problem in a systemic way. The coordination life cycle consists of three states:

* At build-time, a "static interaction map" can be constructed on the relationship of functions and attributes. This map can be used to (pre-)define policies and priorities for identified conflicts.

* At deploy-time, autonomic functions are not yet active/acting on the network. A "dynamic interaction map" is created for each instance of each autonomic function on a per-resource basis, including the actions performed and their relationships. This map provides the basis to identify conflicts that will happen at runtime, categorize them, and plan for the appropriate coordination strategies and mechanisms.

* At runtime, when conflicts happen, arbitration is driven by the coordination strategies. Also, new dependencies can be observed and inferred, resulting in an update of the dynamic interaction map and adaptation of the coordination strategies and mechanisms.

Multiple coordination strategies and mechanisms exist and can be devised. The set ranges from basic approaches (such as random process or token-based process), to approaches based on time separation and hierarchical optimization, to more complex approaches (such as multi-objective optimization and other control theory approaches and algorithm families).

8.2. Coordination Functional Block (*)

A common coordination functional block is a desirable component of the ANIMA reference model. It provides a means to ensure network properties and predictable performance or behavior, such as stability and convergence, in the presence of several interacting autonomic functions.

A common coordination function requires:

* A common description of autonomic functions, their attributes, and life cycle.

* A common representation of information and knowledge (e.g., interaction maps).

* A common "control/command" interface between the coordination "agent" and the autonomic functions.

Guidelines, recommendations, or BCPs can also be provided for aspects pertaining to the coordination strategies and mechanisms.

9. Security Considerations

In this section, we distinguish outsider and insider attacks. In an outsider attack, all network elements and protocols are securely managed and operating, and an outside attacker can sniff packets in transit, inject, and replay packets. In an insider attack, the attacker has access to an autonomic node or other means (e.g., remote code execution in the node by exploiting ACP-independent vulnerabilities in the node platform) to produce arbitrary payloads on the protected ACP channels.

If a system has vulnerabilities in the implementation or operation (configuration), an outside attacker can exploit such vulnerabilities to become an insider attacker.

9.1. Protection against Outsider Attacks

Here, we assume that all systems involved in an Autonomic Network are secured and operated according to best current practices. These protection methods comprise traditional security implementation and operation methods (such as code security, strong randomization algorithms, strong passwords, etc.) as well as mechanisms specific to an Autonomic Network (such as a secured MASA service).

Traditional security methods for both implementation and operation are outside the scope of this document.

AN-specific protocols and methods must also follow traditional security methods, in that all packets that can be sniffed or injected by an outside attacker are:

* protected against modification

* authenticated

* protected against replay attacks

* confidentiality protected (encrypted)

In addition, the AN protocols should be robust against packet drops and man-in-the-middle attacks.

How these requirements are met is covered in the AN Standards Track documents that define the methods used, specifically [RFC8990], [RFC8994], and [RFC8995].

Most AN messages run inside the cryptographically protected ACP. The unprotected AN messages outside the ACP are limited to a simple discovery method, defined in Section 2.5.2 of [RFC8990]: the Discovery Unsolicited Link-Local (DULL) message, with detailed rules on its usage.

If AN messages can be observed by a third party, they might reveal valuable information about network configuration, security precautions in use, individual users, and their traffic patterns. If encrypted, AN messages might still reveal some information via traffic analysis.

9.2. Risk of Insider Attacks

An Autonomic Network consists of autonomic devices that form a distributed self-managing system. Devices within a domain have credentials issued from a common trust anchor and can use them to create mutual trust. This means that any device inside a trust domain can by default use all distributed functions in the entire autonomic domain in a malicious way.

An inside attacker, or an outsider in the presence of protocol vulnerabilities or insecure operation, has the following generic ways to take control of an Autonomic Network:

* Introducing a fake device into the trust domain by subverting the authentication methods. This depends on the correct specification, implementation, and operation of the AN protocols.

* Subverting a device that is already part of a trust domain and modifying its behavior. This threat is not specific to the solution discussed in this document and applies to all network solutions.

* Exploiting potentially yet unknown protocol vulnerabilities in the AN or other protocols. This is also a generic threat that applies to all network solutions.

The above threats are, in principle, comparable to other solutions: in the presence of design, implementation, or operational errors, security is no longer guaranteed. However, the distributed nature of AN, specifically the ACP, increases the threat surface significantly. For example, a compromised device may have full IP reachability to all other devices inside the ACP and can use all AN methods and protocols.

For the next phase of the ANIMA Working Group, it is therefore recommended to introduce a subdomain security model to reduce the attack surface and not expose a full domain to a potential intruder. Furthermore, additional security mechanisms on the ASA level should be considered for high-risk autonomic functions.

10. IANA Considerations

This document has no IANA actions.

11. References

11.1. Normative References

[IDevID] IEEE, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", IEEE 802.1AR, <https://1.ieee802.org/security/802-1ar>.

[RFC8990] Bormann, C., Carpenter, B., Ed., and B. Liu, Ed., "GeneRic Autonomic Signaling Protocol (GRASP)", RFC 8990, DOI 10.17487/RFC8990, May 2021, <https://www.rfc-editor.org/info/rfc8990>.

[RFC8994] Eckert, T., Ed., Behringer, M., Ed., and S. Bjarnason, "An

Autonomic Control Plane (ACP)", RFC 8994, DOI 10.17487/RFC8994, May 2021, <https://www.rfc-editor.org/info/rfc8994>.

[RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <https://www.rfc-editor.org/info/rfc8995>.

11.2. Informative References

[ANIMA-INTENT]
Du, Z., Jiang, S., Nobre, J. C., Ciavaglia, L., and M. Behringer, "ANIMA Intent Policy and Format", Work in Progress, Internet-Draft, draft-du-anima-an-intent-05, 14 February 2017, <https://tools.ietf.org/html/draft-du-anima-an-intent-05>.

[ASA-GUIDELINES]
Carpenter, B., Ciavaglia, L., Jiang, S., and P. Peloso, "Guidelines for Autonomic Service Agents", Work in Progress, Internet-Draft, draft-ietf-anima-asa-guidelines-00, 14 November 2020, <https://tools.ietf.org/html/draft-ietf-anima-asa-guidelines-00>.

[GRASP-DISTRIB]
Liu, B., Ed., Xiao, X., Ed., Hecker, A., Jiang, S., Despotovic, Z., and B. Carpenter, "Information Distribution over GRASP", Work in Progress, Internet-Draft, draft-ietf-anima-grasp-distribution-02, 8 March 2021, <https://tools.ietf.org/html/draft-ietf-anima-grasp-distribution-02>.

[Meyer97] Meyer, B., "Object-Oriented Software Construction (2nd edition)", Prentice Hall, ISBN 978-0136291558, 1997.

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

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

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

[RFC8991] Carpenter, B., Liu, B., Ed., Wang, W., and X. Gong, "GeneRic Autonomic Signaling Protocol Application Program Interface (GRASP API)", RFC 8991, DOI 10.17487/RFC8991, May 2021, <https://www.rfc-editor.org/info/rfc8991>.

[RFC8992] Jiang, S., Ed., Du, Z., Carpenter, B., and Q. Sun, "Autonomic IPv6 Edge Prefix Management in Large-Scale Networks", RFC 8992, DOI 10.17487/RFC8992, May 2021, <https://www.rfc-editor.org/info/rfc8992>.

[SUPA-DATA]
Halpern, J. and J. Strassner, "Generic Policy Data Model for Simplified Use of Policy Abstractions (SUPA)", Work in Progress, Internet-Draft, draft-ietf-supa-generic-policy-data-model-04, 18 June 2017, <https://tools.ietf.org/html/draft-ietf-supa-generic-policy-data-model-04>.

Acknowledgements

The following people provided feedback and input to this document: Sheng Jiang, Roberta Maglione, Jonathan Hansford, Jason Coleman, and Artur Hecker. Useful reviews were made by Joel Halpern, Radia Perlman, Tianran Zhou, and Christian Hopps.

Contributors

Significant contributions to this document were made by John Strassner (Huawei), Bing Liu (Huawei), and Pierre Peloso (Nokia).

Authors' Addresses

Michael H. Behringer (editor)

Email: Michael.H.Behringer@gmail.com

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

Email: brian.e.carpenter@gmail.com

Toerless Eckert
Futurewei USA
2330 Central Expy
Santa Clara, CA 95050
United States of America

Email: tte+ietf@cs.fau.de

Laurent Ciavaglia
Nokia
Villarceaux
91460 Nozay
France

Email: laurent.ciavaglia@nokia.com

JÃ©ferson Campos Nobre
Federal University of Rio Grande do Sul (UFRGS)
Av. Bento GonÃ§alves, 9500
Porto Alegre-RS
91501-970
Brazil

Email: jcnobre@inf.ufrgs.br