ANIMA
Internet-Draft
Intended status: Informational
Expires: April 18, 2016

M. Behringer, Ed.
Cisco Systems
B. Carpenter
Univ. of Auckland
T. Eckert
Cisco
L. Ciavaglia
Alcatel Lucent
B. Liu
Huawei Technologies
J. Nobre
Federal University of Rio Grande do Sul
J. Strassner
Huawei Technologies

October 16, 2015

A Reference Model for Autonomic Networking
draft-behringer-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

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. In section 5 it describes a high level reference model. 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 is a living document and will evolve with the technical solutions developed in the ANIMA WG. Sections marked with (*) do not represent current charter items. While this document must give a long term architectural view, not all functions will be standardized at the same time.

2. The Network View

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

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

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

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

Figure 1: High level view of an Autonomic Network

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

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. Fundamentally, there are two levels inside an autonomic node: the level of Autonomic Service Agents, and the level of the Autonomic Networking Infrastructure, with the former using the services of the latter. Figure 2 illustrates this concept.

```
+------------------------------------------------------------+
|                                                            |
| +-----------+        +------------+        +------------+  |
| | Autonomic |        | Autonomic  |        | Autonomic  |  |
| | Service   |        | Service    |        | Service    |  |
| | Agent 1   |        | Agent 2    |        | Agent 3    |  |
| +-----------+        +------------+        +------------+  |
|       ^                    ^                     ^         |
| -  -  | -  - API level -  -| -  -  -  -  -  -  - |-  -  -  |
|       V                    V                     V         |
|------------------------------------------------------------|
| Autonomic Networking Infrastructure                        |
|    - Data structures (ex: certificates, peer information)  |
|    - Autonomic Control Plane                               |
|    - Autonomic Node Addressing                             |
|    - Discovery, negotiation and synchronisation functions  |
|    - Intent distribution                                   |
|    - Aggregated reporting and feedback loops               |
|    - Routing                                               |
|------------------------------------------------------------|
|             Basic Operating System Functions               |
+------------------------------------------------------------+
```

Figure 2: Model of an autonomic node

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

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

4.1.1. Naming requirements

- Representing each device

  Inside a domain, each autonomic device needs a domain specific identifier.

  [Open Questions] Are there devices that don’t need names? Do ASAs need names?

- Uniqueness

  The names MUST NOT collide within one autonomic domain.

  It is acceptable that the names in different domains collide, since they could be distinguished by domains.

- Semantic Encoding

  It is RECOMMENDED that the names encode some semantics rather than meaningless strings. The semantics might be:

  - Location

  - Device type

  - Functional role

  - Ownership

  - etc.

This is for ease of management consideration that network administrators could easily recognize the device directly through the names.

- Consistency

  The devices’ naming SHOULD follow the same pattern within a domain.

4.1.2. Proposed Mechanisms

- Structured Naming Pattern

  The whole name string could be divided into several fields, each of which representing a specific semantic as described above. For example: Location-DeviceType-FunctionalRole-DistinguisherNumber@NameofDomain.

  The structure should be flexible that some fields are optional. When these optional fields are added, the name could still be recognized as the previous one. In above example, the "DistinguisherNumber" and "NameofDomain" are mandatory whereas others are optional. At initial stage, the devices might be only capable of self-generating the mandatory fields and the "DeviceType" because of the lack of knowledge. Later, they might have learned the "Location" and "FunctionalRole" and added the fields into current name. However, the other devices could still recognize it according to the same "DistinguisherNumber".

- Advertised Common Fields

  Some fields in the structured name might be common among the domain (e.g. "Location" "NameofDomain"). Thus, these part of the names could be advertised through Intent DistributionSection 4.5.

- Self-generated Fields

  The mandatory fields SHOULD be self-generated so that one device could name itself sufficiently without any advertised knowledges.

  There should various methods for a device to extract/generate a proper word for each mandatory semantic fields (e.g. "DeviceType", "DistinguisherNum") from its self-knowledge.

Detailed design of specific naming patterns and methods are out of scope of this document.

4.2. Addressing

Autonomic Service Agents (ASAs) need to communicate with each other, using the autonomic addressing of the node they reside on. This section describes the addressing approach of the Autonomic Networking Infrastructure, used by ASAs. It does NOT describe 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.jiang-auto-addr-management]. The addressing of the Autonomic Networking Infrastructure is in scope for this section, the address space they negotiate for the data plane is not.

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

An autonomic addressing scheme has the following requirements:

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

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

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

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

- Support for virtualization: Autonomic Nodes may support Autonomic Service Agents in different virtual machines or containers. The addressing scheme should support this architecture.

- Simplicity: To make engineering simpler, and to give the human administrator an easy way to trouble-shoot autonomic functions.

- 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 primary use for the autonomically managed addressing described here is for the Autonomic Control Plane ([I-D.ietf-anima-autonomic-control-plane]). The fundamental concepts, as well as the proposed addressing scheme for the ACP is discussed in [I-D.behringer-anima-autonomic-addressing].

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 intent distribution, Section 4.5).

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 Discovery and Negotiation Protocol for Autonomic Networking" [I-D.ietf-anima-grasp] describes more detailed requirements for discovery, negotiation and synchronization in an autonomic network. It also defines a protocol, GDNP, for this purpose, including an integrated but optional discovery protocol.

4.5. Intent Distribution

Intent is the policy language of an Autonomic Network; see Section 8.2 for general information on Intent. The distribution of Intent is also a function of the Autonomic Control Plane. It is expected that Intent will be expressed as quite complex human-readable data structures, and the distribution mechanism must be able to support that. Some Intent items will need to be flooded to most or all nodes, and other items of Intent may only be needed by a few nodes. Various methods could be used to distribute Intent across an autonomic domain. One approach is to treat it like any other technical objective needing to be synchronized across a set of nodes. In that case the autonomic signaling protocol could be used (previous section).

4.6. 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.

4.7. The Autonomic Control Plane

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

5. Functional Overview

This section provides an overview on how the functions in the Autonomic Networking Infrastructure work together, and how the various documents about AN relate to each other.

The foundations of Autonomic Networking, definitions and gap analysis in the context of the IETF are described in [RFC7575] and [RFC7576].

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

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

The adjacency table is fed by the following inputs:

- Link local discovery: This interaction happens in the data plane, using IPv6 link local addressing only, because this addressing type is itself autonomic. This way the nodes learns about all autonomic nodes around itself. This is described in [I-D.ietf-anima-grasp].

- Vendor re-direct: A new device may receive information on where its home network is through a vendor based MASA re-direct; this is typically a routable address. See [I-D.pritikin-bootstrapping-keyinfrastructures].

- Non-autonomic input: A node may be configured manually with an autonomic peer; it could learn about autonomic nodes through DHCP options, DNS, and other non-autonomic mechanisms. Generally such non-autonomic mechansims require some administrator intervention. The key purpose is to by-pass a non-autonomic device or network.

As this pertains to new devices, it is covered in Section 5.3 of [I-D.pritikin-bootstrapping-keyinfrastructures].

The adjacency table is defining the behaviour of an autonomic node:

- If the node has not bootstrapped into a domain (i.e., doesn’t have a domain certificate), it rotates through all nodes in the adjacency table that claim to have a domain, and will attempt bootstrapping through them, one by one. One possible response is a vendor MASA re-direct, which will be entered into the adjacency table (see second bullet above). See [I-D.pritikin-bootstrapping-keyinfrastructures].

- If the node has bootstrapped into a domain (i.e., has a domain certificate), it will act as a proxy for neighboring nodes that need to be bootstrapped. See [I-D.pritikin-bootstrapping-keyinfrastructures].

- If the adjacent node has the same domain, it will authenticate that adjacent node and establish the Autonomic Control Plane (ACP). See [I-D.ietf-anima-autonomic-control-plane].

- 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 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.

6. 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.

A completely autonomic way to automatically and securely deploy such a trust infrastructure is to set up a trust anchor for the domain, and then use an approach as in the document "Bootstrapping Key Infrastructures" [I-D.pritikin-bootstrapping-keyinfrastructures].

6.1. 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.

6.2. Domain Certificate

We need to define how the fields in a domain certificate are to be used. [tbc]

6.3. The MASA

Explain briefly the function, point to [I-D.pritikin-bootstrapping-keyinfrastructures]. [tbc]

6.4. Sub-Domains (*)

Explain how sub-domains are handled. (tbc)

6.5. Cross-Domain Functionality (*)

Explain how trust is handled between different domains. (tbc)

7. Autonomic Service Agents (ASA)

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

7.1. General Description of an ASA

general concepts, such as sitting on top of the ANI, etc. Also needs to explain that on a constrained node not all ASAs may run, so we have two classes of ASAs: Ones that run on an unconstrained node, and limited function ASAs that run also on constrained nodes. We expect unconstrained nodes to support all ASAs.

7.2. Specific ASAs for the Enrolment Process

The following ASAs provide essential, required functionality in an autonomic network, and are therefore mandatory to implement on unconstrained autonomic nodes.

7.2.1. The Enrolment ASA

This section describes the function of an autonomic node to bootstrap into the domain with the help of an enrolment proxy (see previous section). [tbc]

7.2.2. The Enrolment Proxy ASA

This section describes the function of an autonomic node that helps a non-enrolled, adjacent devices to enrol into the domain. [tbc]

7.2.3. The Registrar ASA

This section describes the registrar function in an autonomic network. It explains the tasks of a registrar element, and how registrars are placed in a network, redundancy between several, etc. [tbc]

8. Management and Programmability

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

8.1. 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 [RFC7575].

8.2. Intent (*)

This section describes 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 describe 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. In an ideal autonomic domain, only one intent provided by human administrators is necessary to operate such domain [RFC7576]. However, it is als expected intent definition from autonomic function(s) and even from traditional network management elements (e.g., OSS).

Intent can be refined to lower level policies using different approaches, such as Policy Continuum model [ref]. This is expected in order to adapt the intent to the capabilities of managed devices. In this context, intent may contain role or function information, which can be translated to specific nodes [RFC7575]. One of the possible refinements of the intent is the refinement to Event Condition Action (ECA) rules. Such rules, which are more suitable to individual entities, can be defined using different syntax and semantics.

Different parameters may be configured for intents. 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 (towards intended operational point).

Some examples of parameters for intents are:

- Model version: The version of the model used to define the intent.

- Domain: The network scope in which the intent has effect.

- Name: The name of the intent which describes the intent for human operators.

- Version: The version of the intent, which is primarily used to control intent updates.

- Signature: The signature is used as a security mechanism to provide authentication, integrity, and non-repudiation.

- Timestamp: The timestamp of the creation of the intent using the format supported by the IETF [TBC].

- Lifetime: The lifetime in which the intent may be observed. A special case of the lifetime is the definition of permanent intents.

Intent distribution is considered as one of the common control and management functions of an autonomic network [RFC7575]. Since distribution is fundamental for autonomic networking, it is necessary a mechanism to provision intent by all devices in a domain [I-D.ietf-anima-grasp]. The distribution of Intent is function of the Autonomic Control Plane and several methods can be used to distribute Intent across an autonomic domain [draft-behringer-anima-reference-model]. Intent distribution might not use the ANIMA signaling protocol itself [I-D.ietf-anima-grasp], but there is a proposal to extend such protocol for intent delivery [draft-liu-anima-intent-distribution].

8.3. Aggregated Reporting (*)

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]. The information gathering and the reporting delivery should be done through the autonomic control plane.

Several events can occur in an autonomic network in the same way they can happen in a traditional network. These events can be produced considering traditional network management protocols, such as SNMP and syslog. However, when reporting to a human administrator, such events should be aggregated in order 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 in order to maintain in 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. An alternative to model this is the use of exception-based management [RFC7575].

8.4. 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.

8.5. 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 [draft-strassner-anima-control-loop] 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.

8.6. 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:

- 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

8.7. 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.

9. Coordination Between Autonomic Functions (*)

9.1. 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.

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

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

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

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

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

Multiple coordination strategies and mechanisms 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.

9.2. A Coordination Functional Block (*)

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

A common coordination function requires:

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

- 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.

10. Security Considerations

10.1. Threat Analysis

This is a preliminary outline of a threat analysis, to be expanded and made more specific as the various Autonomic Networking specifications evolve.

Since AN will hand over responsibility for network configuration from humans or centrally established management systems to fully distributed devices, the threat environment is also fully distributed. On the one hand, that means there is no single point of failure to act as an attractive target for bad actors. On the other hand, it means that potentially a single misbehaving autonomic device could launch a widespread attack, by misusing the distributed AN mechanisms. For example, a resource exhaustion attack could be launched by a single device requesting large amounts of that resource from all its peers, on behalf of a non-existent traffic load. Alternatively it could simply send false information to its peers, for example by announcing resource exhaustion when this was not the case. If security properties are managed autonomically, a misbehaving device could attempt a distributed attack by requesting all its peers to reduce security protections in some way. In general, since autonomic devices run without supervision, almost any kind of undesirable management action could in theory be attempted by a misbehaving device.

If it is possible for an unauthorised device to act as an autonomic device, or for a malicious third party to inject messages appearing to come from an autonomic device, all these same risks would apply.

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). AN messages are liable to be exposed to third parties on any unprotected Layer 2 link, and to insider attacks even on protected Layer 2 links.

11. IANA Considerations

This document requests no action by IANA.

12. Acknowledgements

Many people have provided feedback and input to this document: Sheng Jiang, Roberta Maglione, Jonathan Hansford.

13. References

[I-D.behringer-anima-autonomic-addressing]
Behringer, M., "An Autonomic IPv6 Addressing Scheme", draft-behringer-anima-autonomic-addressing-02 (work in progress), October 2015.

[I-D.ietf-anima-autonomic-control-plane]
Behringer, M., Bjarnason, S., BL, B., and T. Eckert, "An Autonomic Control Plane", draft-ietf-anima-autonomic-control-plane-01 (work in progress), October 2015.

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", draft-ietf-anima-grasp-01 (work in progress), October 2015.

[I-D.jiang-auto-addr-management]
Jiang, S., Carpenter, B., and Q. Qiong, "Autonomic Networking Use Case for Auto Address Management", draft-jiang-auto-addr-management-00 (work in progress), April 2014.

[I-D.pritikin-bootstrapping-keyinfrastructures]
Pritikin, M., Behringer, M., and S. Bjarnason, "Bootstrapping Key Infrastructures", draft-pritikin-bootstrapping-keyinfrastructures-01 (work in progress), September 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

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

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

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

Authors’ Addresses

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

Email: mbehring@cisco.com

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

Email: brian.e.carpenter@gmail.com

Toerless Eckert
Cisco

Email: eckert@cisco.com

Laurent Ciavaglia
Alcatel Lucent
Route de Villejust
Nozay 91620
France

Email: laurent.ciavaglia@alcatel-lucent.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