Autonomic Networking Gets Serious

By the ANIMA author team

Introduction

In May 2021, six RFCs about autonomic networking were published[5,6,7,8,9,10]as a result of the work of the "Autonomic Networking Integrated Model and Approach" (ANIMA) working group of the IETF. These RFCs complete the initial charter of that working group, which was started in late 2014 (see [11] for a summary of its inception; however, the first documents to be discussed in the IETF and IRTF were posted in 2012[13]). This foundation now allows the industry to build IETF-standardized network solutions for an "Autonomic Networking Infrastructure" (ANI) into every network device.

This article starts with an overview of the reasoning behind autonomic networking and a description of an early usage scenario. It then gives an overview of the newl

y published specifications and how they will interwork with existing network management, before concluding with several specific use cases.

~~What is this all about?~~ One way to sum ~~it~~ up autonomic networking is "plug and play" for professional networks. This can mean "plug and play for the ISP" or "for the enterprise" or "for industrial networks". This is a significant step forward from the well known idea of plug and play for home networks, which the IETF addresses in the HOMENET WG.

The term "autonomic computing" was coined in 2001~~as early as 20 years ago~~ by IBM. The autonomic nervous system acts largely unconsciously and regulates bodily functions such as heart rate. Autonomic computing was defined by IBM as "self-managing distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity from operators and users." This~~It~~ led naturally to the idea of autonomic networking, which became a topic of discussion and work in the IRTF Network Management Research Group. This resulted in RFCs[1,2] describing the outline of an envisioned autonomic networking infrastructure (ANI) and ultimately in the creation of the ANIMA WG. Since then, various aspects of the problem space were addressed in research, and in proprietary implementations by some vendors. But as always, the need is for interoperability, so proprietary methods have to give way to industry standards. This is the job of the ANIMA working group.

The goal is self-management of networks, including self-configuration, self-optimization, self-healing and self-protection (sometimes collectively called *self-X*). Autonomic Networking (AN) puts operational intelligence into algorithms at the node level, to minimize dependency on human administrators and central management. Nodes capable of AN will discover information about the surrounding network and negotiate parameter settings with their neighbors and other nodes. Later, nodes may also have learning and cognitive capability, i.e. the ability to self-adapt their decision-making process based on information and knowledge sensed from their environment.

Science fiction? Not really. Distributed routing protocols as introduced with the ARPANET in the 1970s and later in the Internet are at their core autonomic: self-configuring, self-optimizing, self-healing. Examples include OSPF (Open Shortest Path First) and IS-IS (Intermediate System to Intermediate System). But over the decades, even those protocols have evolved to become provisioning monsters requiring the human configuration of obscure~~"nerd-knob"~~ parameters and policies ~~for operators~~. A whole industry and research discipline for network Operations Administration and Management (OAM) evolved to define architectures consisting of ~~an~~ ever more complex ~~multitude of~~ layers between the ~~actual~~human intent for the service level objectives of the network (and by implication its protocols) and all th~~os~~e detailed~~"magic"~~ parameters that need to be provisioned consistently and dynamically into each network device whenever there is any change. (As evidence, consider that the IETF alone has published more than 120 YANG modules and sub-modules, each of which contains many individual parameters.)

In today's networks, routing and traffic-engineering parameters are almost exclusively implemented through a centralized set of "Software Defined Networking" (SDN) controller and orchestrator tools configured by human operators. Although a great improvement on older methods, these solutions are still difficult and expensive to build, maintain, validate, predict, secure and above all to make reliable and resilient. These problems are rarely seen from the outside, except when network services are under oversight of regulatory entities that publish reports of those problems, such as[12]. SDN architectures are also highly proprietary, very often from a single vendor, and typically require significant customization through programming for any multi-vendor network deployment. They therefore require network owners to not only hire network operators but also have them become SDN developers. And sometimes, expensive experts have to travel unexpectedly at any hour of the day to fix or update systems. These issues largely arise because of the lack of the automation inside switches and routers that autonomic networking aims to enable.

Nevertheless, these SDN methods are the best option for existing large networks. They are marketed with terms that evolved in the last few years, such as "Zero Touch Networks", "Intent Based Networking", or "Self Driving Networks". In the metaphor of a network being a car, Figure 1 shows how today's networks are driven, and how ANIMA would like them to be.

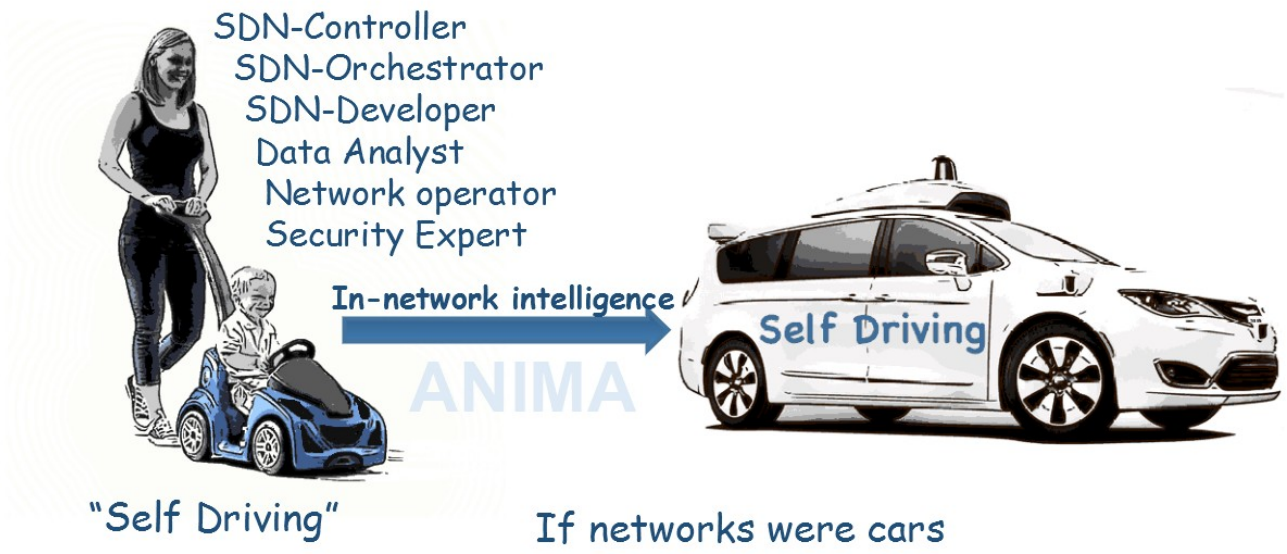

Figure 1: The automobile metaphor

Thee long-term vision for autonomic networkingANIMA is broader than theits newly published standards and short-term standardization goals. cars is rapidly improving driver-assist systems, tmost the near term focus for Much likeThe autonomic networking infrastructure (ANI) as defined in the recent ANIMA RFCs is intended to provide the foundational building blocks. These building blocks are meant to fit seamlessly with existing network and SDN/OAM designs and to improve their metrics such as simplicity, reliability and security. Likewise, the ANI allows designers to more easily embed automation into network devices whenever there is a need. It is worth noting that today, unlike in the past, it is economic to provide enough computing power in network elements to support autonomy.

What can the Autonomic Networking Infrastructure do for You?

Instead of jumping directly into explanation of how the ANI works, welet's first give a simple example of what the operator experience of a typical simple autonomicANI network could be.

In Figure 2-, an operator wants to deploy a new network of devices such as(routers and switches, namely those in the box labeled (2). These devices may be scattered across different physical locations, such as different offices or buildings.). The actual reception of the new, factory fresh equipment, unpacking and physical attachment to pre-existing links may beis performed in different locations by- other personnel who only need to know how to connect power and network cables accurately.

In contrast, without autonomic solutions, this is a very complex, insecure and error-prone process, and the description of all the challenges experienced would be much longer than this article. They may be as simple as connecting a new device into a wrong Ethernet port, whereas any port would work for autonomic bootstrap. Often, an operator must ask the local installer to repeatedly power-cycle a device to activate a new or fixed configuration, which will be automatic in the ANI. In the worst case, the

operator must ask the local installer to perform complex actions such as connecting a laptop to the device and configuring obscure and badly documented features. This can result in bizarre telephone interactions such as the operator asking the installer "Please take a photo of that screen and message it to me."

To avoid this, many device installations nowadays are done by staging. The device is first shipped to a central location where it is pre-configured and secured by expert operators on a trusted network, and then it is re-shipped again to the final deployment location. This is more secure and more predictable, but it is a lot more expensive and slower. Eliminating the need for staging is hence one of the main advantages of the autonomic bootstrap process.

With the ANI, t~~T~~he operator only ~~needs to~~ set~~s~~ up a~~n ANI~~ seed router, called the ANI registrar (1), for example in a Network Operations Center (NOC). The rest is fully automatic and secure, with local installation of new equipment by less expert personnel ("plug in power cable, plug data cable into any free Ethernet port"). The~~is~~ NOC setup consists of only three simple steps:

A. ~~[A]~~ Set up the router (1) as the registrar and assign a name to the ANI.

B. ~~[B]~~ Configure some local port(s) to provide link-layer access to the ANI, to connect management equipment such as a laptop for manual access or an SDN controller.

C. ~~[C]~~ Register the certificate of the registrar with the Manufacturer Authorized Signing Authority (MASA) services of the vendors whose routers and switches are being used in the new network (we will soon describe~~see~~ what that does).

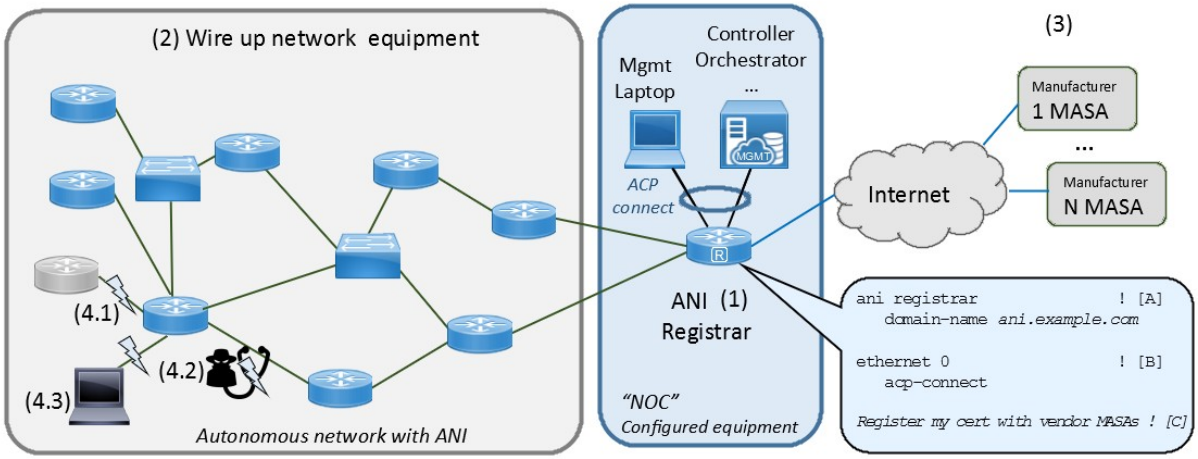

Figure 2: An Example Autonomic Network

Before this seed setup is in place, new routers or switches may be physically interconnected, but they will notdon't do anything. Once they have connectivity to a configured registrar, they will automatically form an ANI as follows.

Each new ANI device (at that stage called a "pledge") will automatically obtain a connection with the ANI registrar and attempt to get enrolled, receiving an ANI certificate so that it can participate. But the registrar first needs to prove to the ANI device that it is its ""owner"". To do that, the registrar communicates (for example over the Internet) with the MASA of the vendor of that device. That MASA has the information that this pledge is actually owned by this registrar's network and returns a security voucher that the registrar can present to the pledge, such that the pledge may now trust the registrar and will therefore accept an ANI certificate from the registrar. This process runs completely automatically without any further handholding or configuration. This part of the ANI is known as Bootstrapping Remote Secure Key Infrastructure[10] (BRSKI, pronounced "Brewski").

Once a new device is enrolled with an ANI certificate, it begins to establish a secure Autonomic Control Plane (ACP) connection with all its neighbors, authenticated and authorized mutually by the devices' ANI certificates. This too happens without any further handholding or configuration. ACP connectivity is always established or re-established between any neighboring ANI routers or switches regardless of any change in topology. It cannot be impacted by faulty operator or SDN configuration of these devices. The goal of the ACP is quite simple: *If there is a physical path to a router or switch, the ACP will automatically provide encrypted and authenticated IPv6 connectivity to it that an operator cannot remove or misconfigure.* This is exactly the type of functionality needed to avoid operational breakdowns such as[12].

Assume all devices were physically connected to each other as shown in Figure 2 and the ANI registrar is connected last (after it was configured). As a result, wWithin minutes, all the devices will have run through BRSKI, and set up the ACP. As a result, the network operator now has secure IP connectivity over the ACP from their management laptop and SDN controller to all ANI devices and can configure them manually or through SDN automation using this connectivity. Each ANI device has a permanent and private IP address within the ANI that does not change, even if the device is physically moved in the network.

But wait! How is this different from 30 year old Ethernet technology? Surely one can simply buy a set of inexpensive Ethernet switches, interconnect them, attach a configuration system at one point and have achieved the same thing?

Indeed, the simplicity of operating Ethernet networks was an inspiration for the ANI, but beyond that, the ANI is fundamentally different. The ANI is above all secure, whereas the default behavior of traditional switches is not. An ANI device can only join the ANI if it is actually owned by the operator, as certified by its manufacturer's MASA, for example via sales records. This means that a stolen device cannot be activated for the ANI in another network. It also means that a device not belonging to this network operator (4.1) cannot be enrolled in an ANI network to launch an attack. To be clear, the operator has not relinquished any control or authority to the manufacturer by this process; only the operator decides which devices may attach to the network and what they may or may not do. The manufacturer's only role is to certify that each device is genuine.

All ACP traffic is encrypted hop-by-hop; therefore all management traffic that uses the ACP, including any legacy unencrypted management protocol, cannot be snooped or spoofed by an attacker (4.2).

Lastly ~~but not least~~, ANI devices, even after having formed the ACP, are still unconfigured, and ideally this means that they should behave like current unconfigured routers: there is nothing running that could provide undesirable network connectivity to any hosts that attach, like some insecure or malicious laptop (4.3). Such an attached device would get no connectivity whatsoever. As a result, there is never a window of opportunity for attackers to impair~~attack~~ unprotected equipment. Instead, the NOC has all the time it needs to remotely provision the devices. In later stages, such provisioning will occur autonomically, as we shall see.

Compared to many other zero-touch solutions, the ANI does not only focus on so-called day-0/day-1 behavior up until the network is operational. Instead its services last through the whole lifecycle. The ANI provides automated certificate renewal for all ANI devices to maintain and refresh its security model. The ACP protects any network OAM traffic that uses it. By its use of hop-by-hop encryption it also continuously protects the whole network and attached OAM equipment from traffic injection or spoofing attacks.

The use of the MASA service is one of crucial benefits of the ANI process to enable reliable and secure device deployment without prior staging. Without a MASA, if an unconfigured device is connected to an unintended or hostile network, it can easily be “kidnapped” by systems that use its default credentials. Furthermore, an attacker could then intercept the enrolment process in order to gain access to the whole network. For a network connection to become hostile, it is often sufficient for some virus impaired device (such as a PC) to be on the same LAN or for the attacker to have impaired other network services such as DNS. Using a MASA to restrict access to cryptographically authorized devices closes off this avenue of attack.

Nevertheless, the MASA concept has raised concerns over the extent of control or observation by the manufacturer. In fact, the MASA can do neither. It can only generate cryptographic vouchers to inform the device who owns it, thereby precluding configuration by anyone else. There are many ways that manufacturers can operationalize this according to their customers’ requirements. The workflow described above, where the owner communicates with the MASA during the device’s enrolment into its owner’s network, is just the simplest option for many owners because it offloads the difficult steps onto the manufacturer.

~~**We now delve into some more technical aspects of the ANIMA solution.**~~

~~Terminology~~

~~Dictionaries differentiate between the terms *automatic*, *autonomous* and *autonomic*:~~

~~*Automatic:* as if done by a machine.~~

~~*Autonomous:* responding and reacting on its own, with no external control.~~

~~*Autonomic:* behaving spontaneously due to internal stimuli.~~

~~The last two are certainly similar, but following industry practice we prefer *autonomic*. The *autonomic nervous system* acts largely unconsciously and regulates bodily functions such as heart rate. *Autonomic computing* was defined by IBM in 2001 as referring to "self-managing distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity from operators and users." We define an *autonomic network* as self-managing (self-configuring, self-protecting, self-healing, self-optimizing) but allowing high-level guidance by a central entity.~~

~~*Autonomic Function:* A specific self-managing feature or function.~~

~~*Autonomic Service Agent (ASA):* An agent that implements an autonomic function, in part (for a distributed function) or whole.~~

~~*Autonomic Node:* A node that embodies autonomic functions.~~

~~*Autonomic Control Plane (ACP):* A self-configuring, fully secure, virtual network used for all autonomic messaging.~~

~~More details about these terms can be found in RFC7575[1] and RFC8993[8].~~

Technical~~O~~ outline of the ANIMA model

As always in network management, there are literally thousands or millions of details that cannot be standardized or even described centrally. What we can do is define a model, a platform, and a toolkit, just as SNMP (Simple Network Management Protocol) and NETCONF (Network Configuration Protocol) have done in the past.

The main terminology we will use is the following. More details about these terms can be found in RFC7575[1] and RFC8993[8]:-

- *Autonomic Function:* A specific self-managing feature or function.
- *Autonomic Service Agent (ASA)*: An agent that implements an autonomic function, in part (for a distributed function) or whole.
- *Autonomic Node:* A node that embodies autonomic functions.
- *Autonomic Control Plane (ACP):* A self-configuring, fully secure, virtual network used for all autonomic messaging.

The main items in the model are:

- Bootstrapping and trust infrastructure[10]. This covers how nodes are authenticated and securely admitted to an autonomic network, and how they establish mutual trust.
- Secure Autonomic Control Plane (ACP)[9]. This is an automatically constructed encrypted virtual network, containing only authenticated nodes that rightfully belong to a particular autonomic domain.
- Discovery for autonomic nodes. This is a mechanism by which nodes attached to the ACP can discover each other. In practice, discovery occurs at a finer grain than nodes, since it really operates at the level of a node's capabilities and objectives.

- Negotiation and synchronization for autonomic nodes. Once nodes have discovered each other, they can synchronize data between themselves, or actively negotiate parameters and resources.
- Autonomic functions operate by negotiating and synchronizing data with their peers in other nodes, and by directly configuring manageable devices in their own scope.
- Discovery, synchronization and negotiation proceed by use of the GeneRic Autonomic Signaling Protocol (GRASP)[5].
- Autonomic service agents (ASAs) are composed of one or more autonomic functions, typically using GRASP via an application programming interface (API)[6].
- Centrally defined policy or configuration rules may be obtained by an ASA via GRASP synchronization, or if appropriate by conventional methods such as an interface to NETCONF or DNS-SD (DNS Service Discovery).

Figure 3 shows an outline of the model as a whole, described in detail in RFC8993[8].

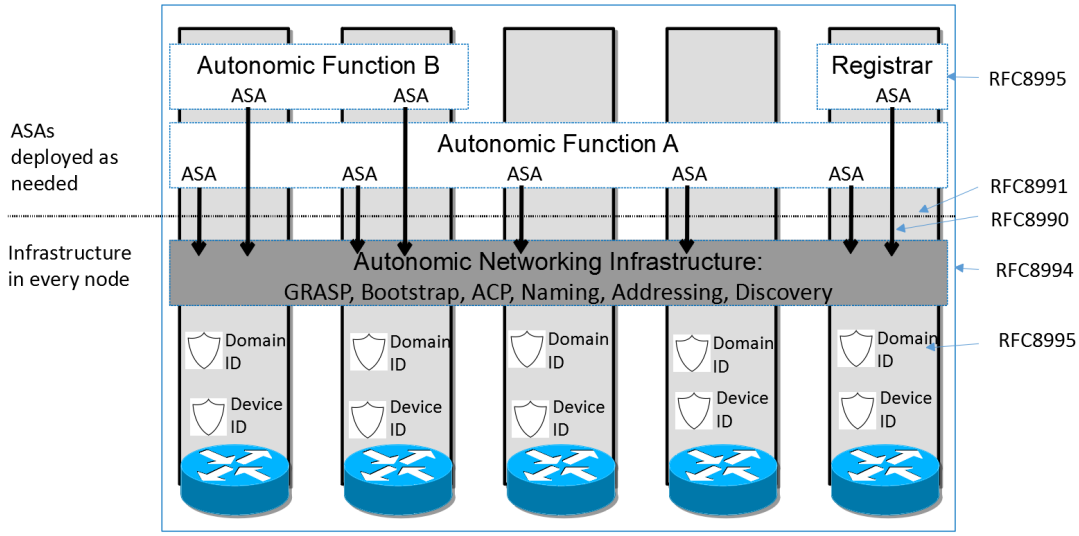

Figure 3: Layered Model of Network with Autonomic Functions

Some Details of Self-configuring Security

~~As mentioned above,~~ ANIMA does not attempt a monolithic bootstrap of a network from a predefined configuration. Instead, it proceeds step by step, and security comes first. The first stage of creating a secure autonomic control plane is bootstrapping a suitable key infrastructure that covers all the nodes that will constitute the ACP. This is done, as previously described, by BRSKI [10]. The process uses manufacturer-installed X.509 certificates (in IEEE 802.1AR IDevID format), in combination with a

manufacturer's authorizing service (MASA). The network administrator decides which devices are authorized to join the network (e.g., by serial number), but relies on the manufacturer to validate each device's certificate whenever the device attempts to join the network via a local "join proxy". These proxies all use a single "domain registrar" node that mediates the authorizing service. The join proxies themselves join the network by the same process; a GRASP mechanism is used for joining nodes (known as "pledges") to find proxies, and for proxies to find each other and the registrar. Only the registrar needs to be configured in advance. ~~(And future work might eliminate even that!)~~

The ACP forms itself among pledges as soon as they have completed their BRSKI enrolment. It is best described as a Virtual Routing and Forwarding (VRF) instance. It is based on a virtual router at each node, consisting of a separate IPv6 forwarding table to which the ACP's virtual interfaces are attached, and an associated IPv6 routing table separate from the data plane. Packet transmission is visible only as IPv6 link-local packets, encapsulating the autonomically created overlay network. This choice was made to ensure that there is no dependency on any pre-existing data plane (either IPv4 or IPv6), because autonomic functions must be able to operate *even if the normal data plane and normal routing are broken.* Even then, the ACP provides a secure channel reaching each node for (re-)configuration, without requiring a physically isolated console port. To start the ACP, all that is required is for each node to create its own IPv6 link-local address on each physical interface, as any modern network device does by default. The VRF consists of point-to-point IPv6 links and is secured using IPsec (IP Security with Internet Key Exchange Protocol Version 2) or DTLS (Datagram Transport Layer Security). From the viewpoint of autonomic service agents, the ACP uses an automatically generated IPv6 Unique Local Address prefix, and it uses RPL (Routing Protocol for Low-Power and Lossy Networks) internally. Like BRSKI, the ACP bootstraps itself, starting with a GRASP-based discovery process.

The security required by the ANI itself is a simple but effective based "group-walled-garden" model for private key infrastructure. It provides strong protection against intruders because of its certificate-based model with automated renewals. It also provides for simple ejection of impaired nodes through certificate revocation, certificate status verification, or short-lived certificates. Further levels of security are easily added when necessary. For example, the ANI itself already uses the common certificate derived role-based security that distinguishes Registrars from other nodes, so that no arbitrary impaired node can overtake the domain by acting as a fake registrar. Such role based security can be expanded to other crucial roles in autonomic functions.

Of course, it would be naive to assume that, even with this key infrastructure and encrypted network, no malicious device, code or user will ever penetrate the autonomic system. A malicious ASA could, for example, attempt a denial of service attack within the ACP. All legitimate ASAs should be designed to take appropriate precautions, and a watchdog ASA could be implemented to detect suspicious activity.

After the secure control plane has configured itself ~~in this way~~, the next stage is to bootstrap connectivity for network management. When this has been achieved, conventional mechanisms (such as an SDN controller) can already reliably and securely reach remote nodes and configure them safely without risk of cutting themselves off. In addition, fully autonomic management mechanisms (i.e.,

ASAs) can start up. To understand how this works, we first need to give more details about the GRASP protocol.

GRASP

GRASP, the GeneRic Autonomic Signaling Protocol[5], is used for signaling between ASAs. These include special-purpose mini-ASAs that support BRSKI (discovery of join proxies and the domain registrar) and ACP creation (discovery of ACP neighbors). Readers will notice that these operations must take place *before* ACP security is in place, so they use a highly restricted subset of GRASP that is limited to specific link-local operations.

After that, GRASP runs over the ACP to guarantee security, so there are no restrictions on allowed operations and any two ASAs in the local domain may trust and communicate with each other. GRASP provides discovery, flooding, synchronization and negotiation mechanisms for the objectives supported by ASAs.

Rather than being a traditional type-length-value protocol, GRASP messages use CBOR (Concise Binary Object Representation), which provides an extensible data model derived from JSON (JavaScript Object Notation), but with a simple and efficient binary encoding. CBOR's flexibility enables GRASP to accommodate a very wide range of data types, with protocol elements often mapping directly into various high-level language representations.

The word "objective" has a special meaning in GRASP. It is a data structure whose main contents are a *name* and a *value*. An objective occurs in three contexts: discovery, negotiation, and synchronization. A single ASA may support multiple independent objectives.

The *name* of an objective is simply a unique string describing its purpose.

The *value* consists of a single configurable parameter or a set of parameters of some kind. The parameter(s) apply to a specific service or function or action. They may in principle be anything that can be set to a specific logical, numerical, or string value, or a more complex data structure. Basically, an objective is defined in the way that best suits its application; that is the great advantage of CBOR encoding. If desired, for example, an objective's *value* could be expressed in the JSON data model. When an objective is shared between ASAs by flooding, synchronization or negotiation, each ASA will maintain its own copy of the objective and its latest value.

GRASP messages allow for *discovery* of an ASA that handles a given objective name; *flooding* a given objective to all ACP nodes (the simplest form of synchronization); *synchronization* of the value of a given objective between two peer ASAs; and *negotiation* of the value of a given objective with a peer ASA.

An Application Programming Interface (API) for GRASP has been defined[6] and implemented as part of a Python 3 prototype. This makes it very easy to implement demonstration ASAs in Python. A partial GRASP implementation has also been made as part of an ACP implementation in the Rust language.

Talking to the NOC

As noted above, a key requirement for the success of ANIMA is smooth integration with existing network management tools and in particular with Network Operations Centers. To this end, an integration mechanism has been documented[4]. The simplest approach is for trusted edge devices in the ACP to "leak" the (otherwise encrypted) ACP natively to certain network management hosts, presumed to be well secured. These edge devices would act as default routers to those management hosts and provide them with IPv6 connectivity into the ACP. A more complex approach would allow the management hosts simultaneous connectivity into the ACP and the traditional data plane.

A related issue is that if the NOC uses DNS Service Discovery (DNS-SD) to announce management services to managed nodes, these announcements will not be automatically available in the ACP, which for security reasons will not have routed access to the data plane where the DNS is available. This again can be solved by a trusted edge device that obtains service information from DNS-SD and redistributes it within the ACP, possibly by the GRASP flooding mechanism. For example, the information for a service named *syslog* could be flooded in a GRASP objective named *SRV.syslog*. Here, the flexibility of CBOR encoding is of great value since a JSON-like structure of service data is common.

Extending that point, since GRASP easily conveys JSON (or practically any other format), it is possible to integrate ASAs communicating via GRASP into almost any part of an existing network management system. For example, an ASA acting as a NETCONF client could retrieve YANG documents from a NOC database via GRASP and the ACP.

Autonomic Function Example 1: Address Management~~of an Autonomic Function~~

A use case that has been fully defined is a GRASP-based mechanism for managing and assigning IP address prefixes[7]. Firstly, we define two GRASP *objectives* for IPv4 or IPv6 prefix management at the edge of large-scale ISP networks. The first objective can be represented thus (in a simplified form):

```
["PrefixManager", [IP_version, prefix_length, prefix]]
```

and the second as

```
["PrefixManager.Params", parameter_info]
```
.

The first objective will be used in GRASP negotiations between two "prefix manager" ASAs in nodes that need to delegate address space to subsidiary routers (using standard IPv6 prefix delegation), when one node is short of spare prefixes and the other one has an adequate pool of unused prefixes. If negotiation succeeds, prefixes will be transferred from one ASA's pool to the other's. If negotiation fails, the ASA that is short of prefixes will use GRASP discovery to find another ASA that can help it. Each participating ASA will require persistent storage to manage its own address pool and to survive power outages or other failures such as network partitions. This will completely obviate any need for human management of an ISP's distributed pool of prefixes, beyond initially configuring the maximum pool in one place.

The second objective may be flooded to all “prefix manager” ASAs to convey relevant policy, which can be enforced during prefix delegation by individual agents. For example, if the flooded parameter information is

```
[
   [["role", "A"],["prefix_length", 34]],
   [["role", "B"],["prefix_length", 44]],
   [["role", "C"],["prefix_length", 56]]
]
```

it would mean that devices of type A are allowed to receive IPv6 prefixes of length 34 bits, and so on.

This mechanism could be used in a variety of ways. One use case is where the three roles above correspond to three functions in an IP Radio Access Network: Radio Network Controller Site Gateways, Aggregation Site Gateways, and Cell Site Gateways. These devices will determine their own roles, and then select the prefix length they are allowed to request and offer to each other accordingly. The only central actions are to define the policy to be flooded out, and to assign the operator’s total address space to a single device that will progressively delegate it to gateways that request prefixes.

We can see from this example that GRASP’s use of CBOR and its easy representation of JSON-like formats gives it great expressiveness and flexibility. While much work remains to be done on individual autonomic functions, the ANI and GRASP provide a solid and flexible foundation for this.

Autonomic Function Example 2: Automating IP Multicast

One common interesting challenge for writing distributed autonomic service agents is solving problems that require decisions about the network topology – in a distributed fashion.

A simple example is automating deployment of a service such as IP Multicast, which needs to determine a small set of designated rendezvous routers, where a key requirement is their location balanced between the center and the edges of a network. Using the ANI and GRASP, it is practical to build such distributed algorithms, for example using common criteria, such as calculation of one’s own average path length as an indicator of centrality, and then running a distributed election algorithm that takes this and other criteria such as node performance and speed of attachment links into account to elect a few top contenders for the role, which then auto-configure the service and their precedence in it.

Autonomic Function Example 3: Automatic protocol security

~~Conclusion: the Operational Role of Autonomic Networking~~

W~~Having looked at one very specific example of an autonomic function, w~~e will end by considering an important early operational role for distributed autonomic behavior. That could start soon with very pragmatic incremental in-network automation, perhaps developed by operators as simple scripts in a scripting language such as Python or Tcl that can run locally on routers.

Consider an existing network where basic services are already running, e.g., IPv4 and/or IPv6 addressing and routing. A software upgrade to the routers that adds support for the ANI could be installed, without impacting any of the pre-existing configuration and services. One of the most

desirable services is protocol security, for example in routing protocols such as OSPF, ISIS and many others. Most protocols have their own security mechanism and/or keying material requirements. However, security is often not configured because there is no automated key management, including key rollover and revocation. Without good automation of key management, networks either fail to enable protocol security, or operators set up a single network-wide password that is never changed. With the ANI, automation of such functionality becomes much simpler, by using GRASP, running securely inside the ACP.

With this in mind, a Python or Tcl script using the GRASP API could easily be written to auto-configure routing protocol security:

- *Discover~~DISCOVER~~* ANI neighbors on links that use the same routing protocol.
- Generate a random key.
- *Negotiate~~NEGOTIATE~~* key with neighbor.
- Configure routing protocol key locally on the router.
- Periodically wake up, *renegotiate~~re-NEGOTIATE~~* and configure a new key.
- Take suitable action if a neighbor disappears or re-appears.

Some protocols may not even have security included in the protocol itself, for example PIM (Protocol Independent Multicast). Instead, packets need to be secured via IPsec security associations (SA). For those protocols, the above script would then auto-configure the IPsec SA instead of an in-protocol key parameter. Such scripts are, of course, autonomic service agents by another name.

In summary, GRASP with ANI can solve the recurring core problems of in-network automation between routers:

> —Q: How to~~do I~~ communicate with a peer (link-local or across other routers) without having any configured IP connectivity?
>
> —A: ACP provides this connectivity automatically with no human intervention.~~.~~
>
> —Q: How to~~do I~~ discover what peers with what type of services there are (especially when not link-local)?
>
> —A: GRASP does this.
>
> —Q: How to~~do I~~ trust these peers?
>
> —A: This trust comes from the ANI certificate used for the ACP. No nodes that have not been registered for the domain and authenticated by their manufacturer can join.
>
> —Q: How to~~do I~~ avoid re-inventing a new protocol to coordinate with ~~my~~ peers ?
>
> —A: Use~~This is what~~ GRASP ~~does~~.

Securing existing protocols is only one example where ANIMA can be put to immediate use. Many or all the benefits apply equally to any other in-network function with similar issues: establishing and

adjusting QoS and other policies, auto-configuring decentralized protocol instances, monitoring, fault isolation and troubleshooting, and even auto-configuring the most basic user network configuration, such as IP prefix distribution as in the previous example. When completely new services are required, ASAs should be developed in languages best suited for such a task.

This immediate applicability to real-world problems provides a significant~~high level of~~ deployment incentive.~~, which will be the basis for ANIMA's bright future.~~

Summary and Conclusion

The ANI is a foundation for network automation. It serves two purposes:

- For existing network OAM designs it provides core functionality to more easily build and deploy networks with secure, resilient network management. ANI provides automated public key deployment and renewal and zero-touch autoconfigured in-band network management connectivity that is protected from being brought down by operator or network management tool errors.
- For ongoing further automation of network OAM (with or without an ultimate goal of fully autonomic networking), the ANI provides fundamental functionality to build distributed, in-network automation agents (ASA) without having to re-implement their core dependencies each time: security, mutual trust, connectivity and network wide and peer-to-peer common signaling (via GRASP).

As a system, ANI may look overwhelming at first with its large set of constituent components (buzzword bingo), but it is fundamentally a very pragmatic approach, with the goal of making network complexity self-managing.

- The basis of ANI is a set of long term well-known and widely used protocol components: IPv6, X.509, IPsec, DTLS, RPL, CBOR, etc.
- The core innovations of ANI are built on top of this foundation: BRSKI, Voucher, MASA on top of X.509, ACP on top of IPsec, DTLS and RPL, and GRASP on top of CBOR.
- ANI is highly modular: All components are defined to be fully reuseable individually or in concert. Only adopt and deploy the subset you need.

References and Further Reading

[1] RFC7575, Autonomic Networking: Definitions and Design Goals. M. Behringer, M. Pritikin, S. Bjarnason, A. Clemm, B. Carpenter, S. Jiang, L. Ciavaglia. June 2015. (DOI: 10.17487/RFC7575)

[2] RFC7576, General Gap Analysis for Autonomic Networking. S. Jiang, B. Carpenter, M. Behringer. June 2015. (DOI: 10.17487/RFC7576)

[3] RFC8366, A Voucher Artifact for Bootstrapping Protocols. K. Watsen, M. Richardson, M. Pritikin, T. Eckert. May 2018. (DOI: 10.17487/RFC8366)

[4] RFC8368, Using an Autonomic Control Plane for Stable Connectivity of Network Operations, Administration, and Maintenance (OAM). T. Eckert, Ed., M. Behringer. May 2018. (DOI: 10.17487/RFC8368)

[5] RFC8990, GeneRic Autonomic Signaling Protocol (GRASP). C. Bormann, B. Carpenter, Ed., B. Liu, Ed. May 2021. (DOI: 10.17487/RFC8990)

[6] RFC8991, GeneRic Autonomic Signaling Protocol Application Program Interface (GRASP API). B. Carpenter, B. Liu, Ed., W. Wang, X. Gong. May 2021. (DOI: 10.17487/RFC8991)

[7] RFC8992, Autonomic IPv6 Edge Prefix Management in Large-Scale Networks. S. Jiang, Ed., Z. Du, B. Carpenter, Q. Sun. May 2021. (DOI: 10.17487/RFC8992)

[8] RFC8993, A Reference Model for Autonomic Networking. M. Behringer, Ed., B. Carpenter, T. Eckert, L. Ciavaglia, J. Nobre. May 2021. (DOI: 10.17487/RFC8993)

[9] RFC8994, An Autonomic Control Plane (ACP). T. Eckert, Ed., M. Behringer, Ed., S. Bjarnason. May 2021. (DOI: 10.17487/RFC8994)

[10] RFC8995, Bootstrapping Remote Secure Key Infrastructure (BRSKI). M. Pritikin, M. Richardson, T. Eckert, M. Behringer, K. Watsen. May 2021. (DOI: 10.17487/RFC8995)

[11] B. Carpenter, Autonomic Networking, IETF Journal, 2014. https://www.ietfjournal.org/autonomic-networking/

[12] FCC, June 15, 2020 T-Mobile Network Outage Report, A Report of the Public Safety and Homeland Security Bureau Federal Communications Commission, PS Docket No. 20-183, October 2020, https://docs.fcc.gov/public/attachments/DOC-367699A1.docx

[13] M. Behringer et al, https://datatracker.ietf.org/doc/html/draft-behringer-autonomic-network-framework-00 and https://datatracker.ietf.org/doc/html/draft-behringer-homenet-trust-bootstrap-00, 2012.

THE ANIMA AUTHOR TEAM is a group of participants in the IETF's ANIMA Working Group, including Michael Behringer, Carsten Bormann, Brian E. Carpenter, Toerless Eckert, Sheng Jiang, Yizhou Li, Jéferson Campos Nobre and Michael Richardson. They may be contacted at anima@ietf.org.