ANA Project

Autonomic Network Architecture

autonomic network architecture

Sixth Framework Programme

Priority FP6-2004-IST-4

Situated and Autonomic Communications (SAC)

Project Number: FP6-IST-27489

Deliverable D1.4/5/6_v1.1

ANA Blueprint – First Version Updated

ANA Blueprint Version 2.0

ANA Project

Autonomic Network Architecture

ana
autonomic network architecture

Project Number	FP6-IST-27489
Project Name	ANA - Autonomic Network Architecture
Document Number	FP6-IST-27489/WP1/D1.4/5/6_v1.1
Document Title	ANA Blueprint – First Version Updated
Workpackage	WP1
Editors	Christophe Jelger (UBasel) Stefan Schmid (NEC)
Authors	Ghazi Bouabene (UBasel) Christophe Jelger (UBasel) Stefan Schmid (NEC)
Reviewers	David Hutchison
Dissemination level	Public
Contractual delivery date	31st December 2008
Delivery Date	15th February 2009
ANA Blueprint Version	2.0

Abstract:

This document is the second version of the ANA Blueprint Architecture. It describes the design and communication model of the Autonomic Network Architecture (ANA) developed within the EU FP6 IST Project 27489.

The Blueprint includes a definition of the basic abstractions, core concepts and communication paradigms of ANA, and it also defines the abstract interfaces that an implementation of a network node must provide in order to support the ANA architectural concepts.

Keywords:

ANA Blueprint Architecture, Basic Abstractions, Communication Paradigms, ANA Node.

Executive Summary

The goal of the ANA project is to explore novel ways of designing and building networks beyond legacy Internet technology. The ultimate goal is to design and develop a novel network architecture that can demonstrate the feasibility and properties of autonomic networking. As specified in the initial description of work of the project, the goal of the project is to develop an innovative network architecture that facilitates the development of autonomic networking features such as self-configuration, self-optimization, self-monitoring, self-management, self-repair, and self-protection.

Because autonomic networking is a relatively new research area and there does not yet exist any autonomic network architecture, the ANA project is doing significant pioneering work on this wide and challenging research topic.

Designing a network architecture implies identifying and specifying the design principles of the system being developed. The architecture hence defines the atomic functions and entities that compose the network and specifies the interactions that occur between these various building blocks.

As the second version of the ANA architectural Blueprint specification, this document describes the updated reference model of the ANA autonomic network architecture. It includes a definition of the basic abstractions and concepts of the ANA architecture and the communication paradigms. The document also includes the abstract interfaces that an implementation of a network node must provide in order to support the ANA architectural concepts.

Revision History:

31.12.06 (M12)	Version 1.0	First draft of the ANA Blueprint Document
31.12.07 (M24)	Version 1.1	Update of the First ANA Blueprint
31.12.08 (M36)	Version 2.0	Update of the ANA Blueprint - Version 2.0 Summary of changes: • Fully re-organized sections 2 and 3 presenting the basic abstractions of ANA and the communication API. • Revised section 4 according to changes in terminology and based on the experience gained in 2008. • Revised section 5 according to the experience gained during

		the implementation phase in 2008. • Removed all the content which is now described in detail in other dedicated Deliverables.

Table of Contents

1 INTRODUCTION

This document is the second version of the ANA Blueprint. It is based on the first versions (i.e. 1.0 and 1.1) of the document which were respectively produced at months M12 and M24 of the project, but it also contains new additions inline with the development of the project. Moreover, some of the content that appeared in the previous versions has been removed since it is now described in more detail in dedicated deliverables on those topics. Version 2.0 of the Blueprint thus focuses only on the core design specifications of ANA.

This update of the ANA Blueprint also reflects the lessons learned from the ongoing prototyping efforts of the ANA core system during the third year (2008) of the project. As the development of the ANA software is still ongoing and maturing, a final Blueprint document will be released at the end of 2009 (M48), which will reflect all the experiences gained during the ANA development from 2006 to 2009.

As for the previous versions, this version of the Blueprint describes the current reference model of the ANA autonomic network architecture. This includes a definition of the basic abstractions and concepts of the ANA architecture along with the main communication paradigms and mechanisms. The Blueprint also includes the abstract interfaces that an implementation of a network node must provide in order to support the ANA architectural concepts.

1.1 Motivation and Aims

This section recaps the main drivers behind the Autonomic Network Architecture (ANA) and its high-level aims.

The overall objective is to develop a novel network architecture and to populate it with the functionality needed to demonstrate the feasibility of autonomic networking. In order to avoid the same shortcomings of network architectures developed in the past, the guiding principles behind the architectural work in ANA are achieving maximum *flexibility* and providing support for *functional scaling* by design.

The former indicates that the project members do not envision a "one-size-fits-all" network architecture for the various different types of networking scenarios (e.g., Internet, Sensor Networks, Mobile Ad hoc Networks, Peer-to-peer Networks). Instead, the aim of ANA is to provide an architectural framework that enables the co-existence and interworking between different "networking styles".

Functional scaling means that a network is able to extend both horizontally (more functionality) as well as vertically (different ways of integrating abundant functionality). New networking functions must be integrated into the "core machinery" of the system, and not on top as "patches"; otherwise we do not have scaling of functionality, only function accumulation on top of an ossified network core. In the context of ANA, the ability of functional scaling is considered vital, as this provides the basis to allow dynamic integration of new, *autonomic* functionalities, as they are developed.

The Internet and the OSI reference model are examples of a functionally non-scalable approach: both are based on the premise of factoring out functionality with a pre-defined and static model. The "canonical set" of protocols collected in the Internet-Suite has done a good job so far; however the limitations of this one-size-fits-all approach have become visible. This is not surprising, since the networking layer of the Internet has not aimed at functional scaling by design: it has evolved through a 'patchwork' style. Additions were made in a stealth-like way and have not dared to change the core of IP networking. The main examples are the introduction of the hidden routing hierarchy with AS, CIDR or MPLS, as well as other less successful initiatives such as RSVP, multicast, mobile IP and MANET.

As a result, the objective of the ANA project is to provide an architectural framework – a "meta architecture" – that allows the accommodation of and interworking between the full range of networks, ranging from small scale Personal Area Networks, through (Mobile) Ad hoc Networks and special purpose networks such as Sensor Networks, to global scale networks, in particular the Internet.

An important step towards such a "meta architecture" is to design a framework that is able to host different types of networks. As a result, ANA encompasses the concept of *compartments* as a key abstraction that allows co-existence and interworking of different types of network through a minimum generic interface.

Furthermore, the operation of different types of compartment must be analysed in order to identify the fundamental building blocks of the abstraction. The decomposition of compartments into the core functions helps to understand how the necessary flexibility and functional scalability to provide *autonomicity* can be achieved through compartments.

1.2 Scope of Deliverable

The Blueprint document focuses on the overall architectural aspects of ANA. As such, it defines the basic abstractions and building blocks of ANA and presents its basic operation and mechanisms. Therefore, it does not capture the details of the many developments going on in other areas of the project, such as intra and inter-compartment routing, service discovery, functional composition, monitoring, etc. At the end of 2008, the outcomes of these activities are described in the following deliverables:

- D1.11: Second public release of the core ANA software
- D2.9: Design and implementation of an intra-compartment routing scheme
- D2.10: Implementation of inter-compartment communication schemes
- D2.12: Integration of the functional composition framework and evaluation
- D2.13: Service discovery phases: conceptual design and evaluation

- D2.14: Implementation of self-association and self-organization mechanisms
- D2.15: Implementation and integration of the proof-of-concept overlay streaming compartment
- D2.16: Final design, evaluation and prototype implementation of content-based routing mechanism
- D3.7: The monitoring part of the ANA architecture - version 2
- D3.8: Self-optimization mechanisms
- D3.9: Implementation of selected components of the failure detection and fault management
- D3.10: Measurement-based resilience mechanisms
- D4.5: Extended integrated test-bed infrastructure
- D4.7: Description of VoD demonstrator design and composition
- D4.8: Integration guidelines for the final ANA distribution
- D4.9: Test environment and guidelines for ANA application and functional block tests

1.3 Structure of the document

This document is organized as follows. Section 2 describes the core abstractions and basic communication paradigms of the ANA network architecture. Section 3 introduces the ANA Compartment API which is used to wrap different "network types" with a generic interface. Section 4 provides details on the core ANA concept of compartment and describes how this is used at the node-level and at the scale of networks. Section 5 describes the ANA Node, its basic components and operation as well as the components that an implementation should support. Finally, Section 6 concludes the document.

1.4 Abbreviations

ANA	Autonomic Network Architecture
API	Application Programming Interface
BP	Bootstrap Protocol
FB	Functional Block
IC	Information Channel
IDP	Information Dispatch Point
IDT	Information Dispatch Table
MC	MINMEX Controller

MINMEX	Minimal INfrastructure for Maximal EXtensibility

2 Core Architectural Abstractions

This section introduces the basic abstractions of ANA: the functional block (FB), the information dispatch point (IDP), the compartment, and the information channel (IC).

2.1 Basic abstractions

2.1.1 Functional Block (FB)

In ANA, any protocol entity generating, consuming, processing and forwarding information is abstracted as a functional block (FB). For example, an IP stack, a TCP module, but also an encryption function or a network monitoring module can be abstracted as a functional block. In contrast to OSI's protocol entities [1], functional blocks are not restricted to being abstractions of network protocols and the functionality they provide can range from a full monolithic network stack down to a small entity that computes checksums (like elements in Click [2]).

The fact that a FB can represent the whole range from an individual processing function to a whole network stack makes this abstraction very useful. It permits the capture and abstraction of many different flavours of communication entities and types, and enables them to interact with other entities inside ANA under a generic form.

In all the figures of all the documents describing ANA and its components, a functional block (FB) is represented by a (red) square as shown in Figure 2-1.

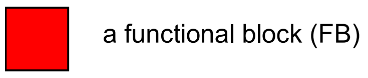

Figure 2-1. Functional Block

2.1.2 Information Dispatch Point (IDP)

The fundamental concept introduced by ANA and around which the core architecture machinery is designed is the information dispatch point (IDP). IDPs are inspired by network pointers [3] and by network indirection mechanisms [4], and they are somehow similar to file descriptors in Unix systems. Basically inside an ANA node, a functional block (FB) is always accessed via one or multiple IDPs attached to it: in other words, all interactions are carried out via an indirection level *built into* the network architecture. In the same way as Unix systems enforce access to all system components (e.g. devices, sockets, files) via file descriptors, ANA mandates that all functional blocks are accessed via IDPs. However unlike file descriptors and sockets, the binding of an IDP (i.e. the FB to which it is attached) is dynamic and can change over time (e.g. if a “network stack” is reconfigured). The bindings between IDPs and FBs are stored in the core forwarding table of the ANA Node where each IDP is identified by a *node-local label* (e.g. a 32-bit unsigned integer value). As detailed later, each IDP also maintains some dynamic information such as the FB to which it is attached, state to be passed to the FB, a lifetime counter, and some status information.

The advantage of IDPs is two-fold: first, they act as generic *communication pivots* between the various functional blocks running inside an ANA node and, second, they provide the required flexibility allowing for the re-organization of communication paths. For packet forwarding, IDPs permit the implementation of forwarding tables which are fully decoupled from addresses and names: i.e., in ANA the next hop “entity” (local or remote) is always identified by an IDP. This permits the easy addition and use of new networking technology and protocols as long as they export their communication services as IDPs.

In all the figures of all the documents describing ANA and its components, an information dispatch point (IDP) is represented by a black dot as shown below. When an IDP is bound to a functional block (FB), the binding is represented by a line connecting the IDP to the FB. Essentially, this means that any data sent to, or any action applied to an IDP is really received or captured by the functional block it is attached to. ANA strictly enforces this level of indirection: that is, it is *not* possible to interact directly with a functional block. All interactions are *always* handled via an IDP. In Figure 2-2, the action of sending to or acting on an IDP is drawn as an arrow.

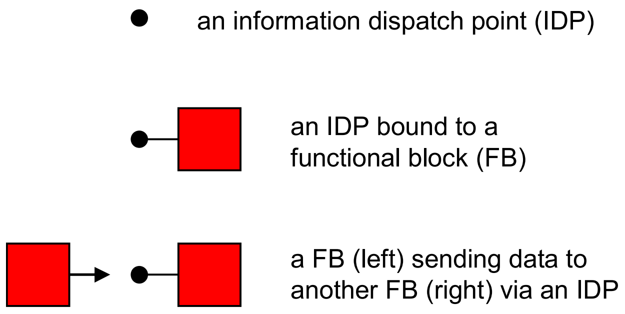
an information dispatch point (IDP)
an IDP bound to a functional block (FB)
a FB (left) sending data to another FB (right) via an IDP

Figure 2-2. Information Dispatch Points

As shown in the Figure 2-3, a given functional block can have multiple IDPs attached to it. Inside the FB, each IDP is typically linked with a specific action, with potentially some specific state attached to it. For example in the figure, the IDP labelled `a` is “bound” to the function `hash_MD5()` while the IDP `b` is attached to the function `hash_SHA()`

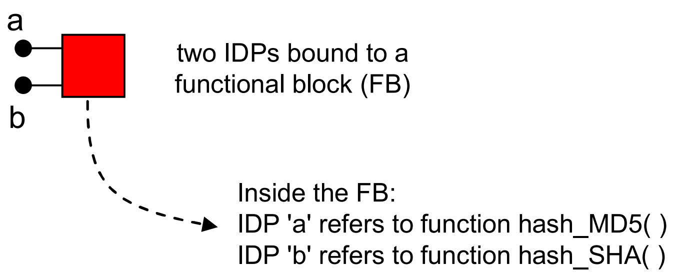
a
b
two IDPs bound to a functional block (FB)
Inside the FB:
IDP 'a' refers to function hash_MD5( )
IDP 'b' refers to function hash_SHA( )

Figure 2-3. IDPs and bindings inside functional blocks

Different IDPs attached to the same functional block can also be bound to the same function but with different state attached to them. For example in Figure 2-4, the IDP labelled `c` is bound to the function `sendto()` with state (argument) `1.2.3.4` while the IDP `d` is attached to the same function with state `9.8.7.6`.

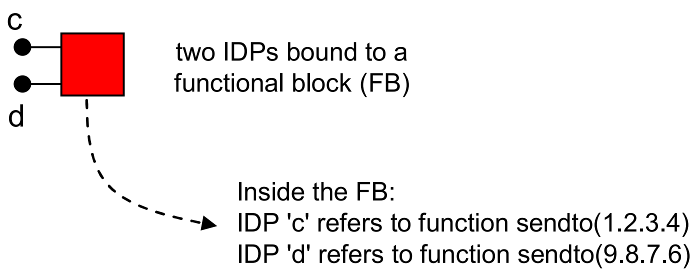
c
d
two IDPs bound to a functional block (FB)
Inside the FB:
IDP 'c' refers to function sendto(1.2.3.4)
IDP 'd' refers to function sendto(9.8.7.6)

Figure 2-4. IDPs and bindings inside functional blocks

2.2 Compartment and Information Channel (IC)

2.2.1 Background and Motivation

Recent research has led to the conclusion that today's networks have to deal with a growing diversity of commercial, social and governmental interests that have led to increasingly conflicting requirements among the competing stakeholders. Clark et al. refer to this development as "tussles in cyberspace" [5]. A pragmatic way to resolve those tussles is to divide the network into different realms [6] or turfs [7] that isolate the conflicting or competing interests.

In order to abstract the many notions of network realms and domains, ANA introduces the key concept of *compartment*. With respect to communication networks, the concept of compartment is similar to the notion of *context* as proposed in Plutarch [8] where "a context describes a region of the network that is homogeneous in some regard with respect to addresses, packet formats, transport protocols, naming services, etc.”

The compartment abstraction, as one of the fundamental concepts of ANA, enables ANA to “encapsulate” (or “wrap”) today’s networks into a *generic* abstraction. Compartments indeed offer a set of generic “wrapper primitives” that allows accommodation of both legacy network technologies and new communication paradigms within ANA in a generic manner. Since support for backwards compatibility and inter-working with already deployed networks is a key requirement for any new network architecture, some architectural compromises (e.g. number of primitives and arguments of the API) are typically made to achieve this. However, as the compartment abstraction permits hiding the internals and specifics of individual network technologies, this concept enables inter-working among heterogeneous compartments in a generic manner.

2.2.2 ANA Compartment: Definition

Prior to defining compartments and their properties, it is important to note that the notion of compartment does not necessarily involve multiple nodes: a compartment can indeed involve a single node only, for example to provide node-local communications between software entities. Hence in the rest of this document, we refer to *network compartment* when multiple nodes are involved, while we will refer to the *node compartment* to describe a specific construct of ANA allowing access to node-local software entities via the set of compartment generic primitives.

In ANA, compartments implement the operational rules and administrative policies for a given communication context. The boundaries of a communication context, and hence the compartment boundaries, are based on technological and/or administrative boundaries. For example, compartment boundaries can be defined by a certain type of network technology (e.g., a specific wireless access network) or based on a particular protocol and/or addressing space (e.g., an IPv4 or and IPv6 network), but also based on a policy domain (e.g., a national health network that requires a highly secure boundary). ANA anticipates that many compartments co-exist and that compartments are able to interwork on various levels.

The complexity and details of the internal operation are left to each compartment. That is, compartments are free to choose internally the types of addressing, naming, routing, networking mechanisms, protocols, packet formats, etc. For example in ANA, typical network compartments are: an Ethernet segment, the public IPv4 Internet, a private IPv4 subnet, the DNS, peer-to-peer systems like Skype, and distributed web caching networks like Akamai.

Typically, the communicating entities inside a compartment are represented in the ANA architecture through functional blocks (FBs). They implement the functionality that is required to communicate within the compartment. As such, the FBs can also be regarded as the processing elements or functions hosted by an ANA node that constitute the “compartment stack”.

2.2.3 Information Channel (IC)

Typically in each network compartment, a distributed set of functional blocks collaborates in order to provide communication services to other compartments and applications. Whatever the nature of the communication service (unicast, multicast, datagram, reliable, etc.), it is abstracted by an *information channel* (IC) which (as anything else in ANA) is accessed via an IDP. While such an IDP is really bound to a functional block belonging to the network compartment, the abstraction provided is an information channel to some remote entity or entities. Note that the end-point of an information channel is not necessarily a network node: it can be a distributed set of nodes or servers, a software module, a routing system, etc. This rather abstract definition allows us to capture and abstract many different flavours of communication forms.

Information Channels (ICs) can be of either a physical or logical nature. Examples of physical ICs are a cable, a radio medium, or memory. A logical (or virtual) IC can represent a chain of packet processing elements and further ICs. The IC abstraction captures various types of communication channels, ranging from point-to-point links or connections, over multicast or broadcast trees to special types of channels such as anycast or concast trees. Figure 2-5 illustrates a number of different types of IC. Note that broadcast is considered a special case of multicast.

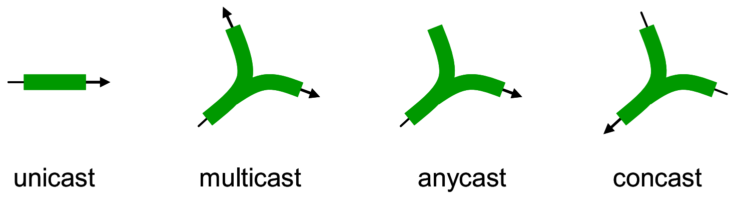

Figure 2-5. Different Types of Information Channel (IC)

The following example (see Figure 2-6) shows how FBs, IDPs and ICs interact in order to enable communication within a compartment. The FBs in this example perform some type of processing (e.g., add protocol header, perform rate control, etc.) on the data, before it is transmitted to an entity in another ANA Node via the IC u → v.

The IC u → v in this example represents the service that is provided by some underlying system (e.g., the operating system providing a link and physical layer functionality). Since this IC is not provided by the network compartment considered in this example, it is shown in the *imported view* of the compartment. As discussed later in section 4.2.2, this IC can also represent the service provided by another, underlying compartment.

The IC s → t shows the "communication service" created by the network compartment as it would appear to an external entity using it. This IC hence appears in the *exported view* of the compartment: typically, the ICs exported by a compartment can be imported by other compartments.

Finally, the *structural view* of the figure shows the internal processing elements (i.e. FBs) that execute the protocols and functions that form the “compartment stack”. Note that for simplicity the figure shows a single FB but in practice there could be multiple interconnected FBs providing the “compartment stack”.

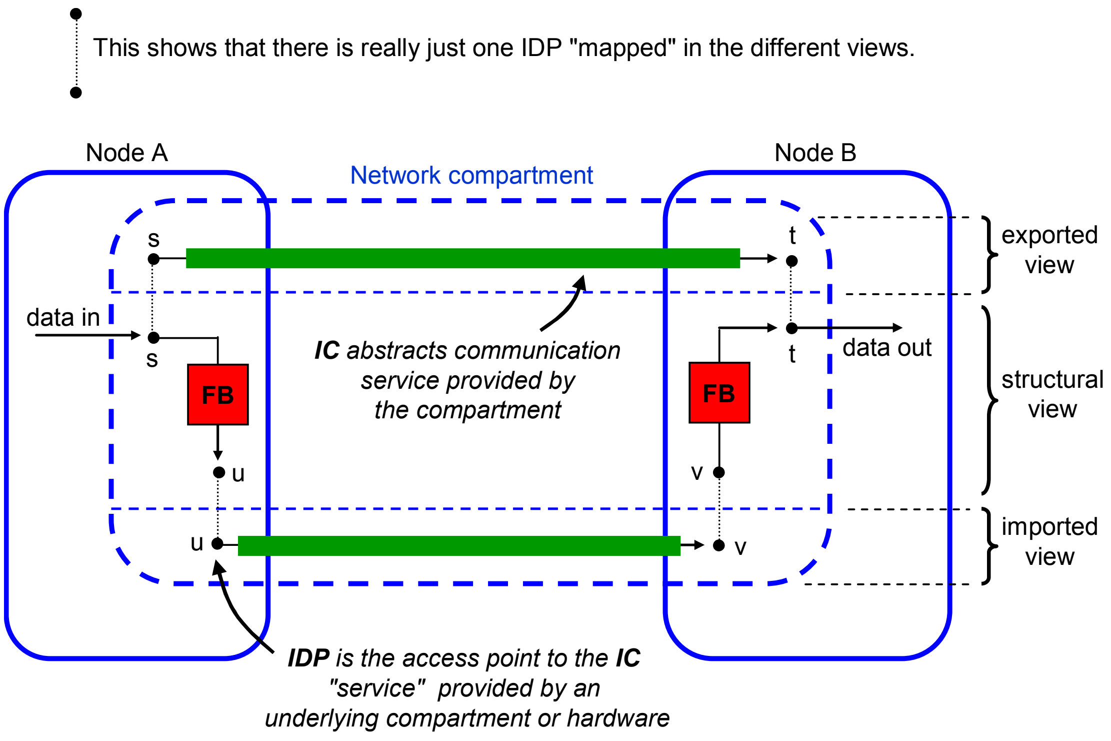
This shows that there is really just one IDP "mapped" in the different views.
Node A
Network compartment
Node B
s
t
exported view
data in
s
t
data out
IC abstracts communication service provided by the compartment
FB
FB
structural view
u
v
u
v
imported view
IDP is the access point to the IC "service" provided by an underlying compartment or hardware

Figure 2-6. Different Views of a Compartment: Imported View, Structural View and Exported View

Figure 2-7 illustrates how compartments are formed by a set of communication elements (FBs), the communication media between the communication elements (ICs), and the dynamic binding elements (IDPs). The figure also shows how an IC can actually abstract (or hide) the service provided by another underlay compartment.

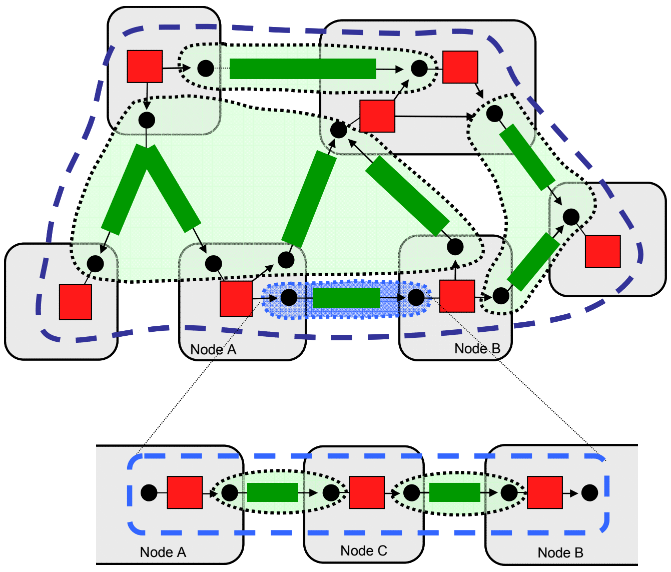

Figure 2-7. The Compartment: A policed set of Functional Blocks (red), Information Channels (green), and Information Dispatch Points (black)

3 THE COMPARTMENT API

As described previously, network compartments in ANA are free to *internally* use their choice of addressing, naming, routing mechanisms, protocols, packet formats, etc. The objective is that ANA fosters variability and evolution by not imposing any restriction on the internal network operation of compartments.

However, to foster and guarantee all possible and unforeseen interactions within ANA, all compartments must support a generic *compartment API* which provides the "glue" that permits the building of complex network stacks and packet processing paths. With respect to application development, a key requirement is that the API supports generic function prototypes and arguments such that applications can use the same programming primitives when communicating to *any* compartment. In other words, a long-term objective of ANA is that developers can write "compartment-agnostic" applications that do not need to be modified to support new network compartments.

For example, with ANA one could write a web browser that takes any *opaque* string as input, finds the right network compartment for handling this address or name, requires from this compartment an information channel to the destination and eventually sends the data to the destination FB without, at any time, having to understand the syntax of the input string. This is different from the current situation with sockets where a programmer must use and properly set the appropriate `sockaddr_` structure depending on the "network compartment" being accessed. This hard-coding assumes prior knowledge of "who uses what" and seriously hinders innovation (e.g. all applications had to be re-written to support IP version 6).

3.1 Basic Primitives

The ANA compartment API currently offers six fundamental primitives detailed below with some simplified C-style function prototypes. . These primitives constitute the fundamental and generic interface used to interact with network compartments. In ANA, all compartments must support these generic primitives.

- **IDP_s = publish (IDP_c, CONTEXT, SERVICE)**: to request from a compartment that a certain *service* becomes reachable via the compartment (identified by IDP_c) in a certain *context*. When successful, `publish` returns an IDP (IDP_s) via which the service has been published.
- **int = unpublish (IDP_c, IDP_s)**: to request from a compartment (identified by IDP_c) that a certain *service* is no longer reachable via this compartment and *context*. The primitive returns an integer value to indicate a successful request or some error condition.

- **IDP_r = resolve (IDP_c, CONTEXT, SERVICE)**: to obtain from a compartment (identified by IDP_c) an information channel to communicate with a certain *service* that has been published in some *context* inside the compartment. When successful, `resolve` returns an IDP (IDP_r) via which the remote service can be reached.
- **int = release (IDP_c, IDP_r)**: to release a previously resolved information channel identified by the IDP_r. The primitive returns an integer value to indicate a successful request or some error condition.
- **info = lookup (IDP_c, CONTEXT, SERVICE)**: to obtain further information from a compartment (identified by IDP_c) about a certain *service* that has been published in some *context* inside the compartment. When successful, `lookup` returns some information (to be detailed later) about the service being looked up.
- **int = send (IDP_r, data)**: to send data to a *service* which has been previously resolved into the IDP_r. The function returns an integer value to indicate a successful request or some error condition.

The two fundamental notions of CONTEXT and SERVICE have been introduced in order to be able to explicitly specify *what* (service) and *where* (context) "inside a compartment" a certain resource must be published or resolved. In other words, the context typically provides a hint to the compartment how to resolve the requested service while the service describes a resource inside the context. These notions are further detailed in the next section.

Note that all compartments such as Ethernet, IP, DNS, etc., must support this generic API. In our current implementation, both the context and service arguments are simply character strings. As such, one can write "compartment-agnostic" code where there is no need to have any a priori knowledge about a certain compartment in order to use its services. This contrasts to the socket API where one must explicitly know which sockaddr data structure should be used with a given "compartment": i.e., `sockaddr_in` for AF_INET, `sockaddr_ll` for AF_PACKET, etc.

3.2 The Context and Service Arguments

The intuitive "design philosophy" behind the ANA API is that, for any kind of network applications and protocols, a user typically wants to find or reach something (the SERVICE) with respect to some scope (the CONTEXT) inside a network compartment. In order words, the SERVICE is what the user is looking for (or wants to communicate with, download, etc.), and the CONTEXT defines where and how (in which scope) the network operation (e.g. resolve) must be done. Based on our current experience with the API, we believe that it is possible to fit most if not all networking applications and protocols into the CONTEXT/SERVICE communication model.

For example, in an IP compartment one would use IP addresses as CONTEXT values with SERVICEs such as `"tcp:80"` or `"udp:53"`, while in a DNS compartment the CONTEXT would be used to specify record types (e.g., `"A"`, `"MX"`) with SERVICEs being FQDNs such as `"www.example.com"`. In a peer-to-peer file sharing system, one could use the CONTEXT to specify a search scope with boolean operators (e.g. `"title AND (rock OR blues)"`) while the SERVICE could specify keywords (e.g. `"satisfaction"`). Note that the SERVICE argument is a description of the service to be published or resolved, and is typically not interpreted by the compartment FB. The type of the SERVICE argument is not fixed: this argument can be a port or protocol number, a string (e.g., URL or filename), an IP address, a hash value, etc.

Finally, ANA also mandates that all network compartments understand two generic CONTEXT values. The generic CONTEXT `"*"` specifies the largest possible scope as interpreted by the compartment, while the `"."` CONTEXT restricts the scope to node-local operations. For example, the `"*"` CONTEXT would map into the CONTEXTs `"FF:FF:FF:FF:FF:FF"` and `"255.255.255.255"` for respectively the Ethernet and IP compartments, while the `"."` CONTEXT would map into the CONTEXTs `"00:00:00:00:00:00"` and `"127.0.0.1"`.

In the current implementation of ANA, the CONTEXT and SERVICE arguments of the API are formed from a simple `<pointer,length>` tuple so there is maximum flexibility to specify them. However, we currently favour printable (human readable) strings as a *public/external* format for both the CONTEXT and SERVICE arguments. The key motivation is that ASCII is a universal format that can be read and given by human users and can easily be manipulated with lexical tools and algebras (e.g. regular expression matching). Moreover, using strings as public formats avoid the need to know anything about the internal encoding used by compartments (e.g. no need to know that an IP address must be encoded in network order).

In the Appendix, we provide some further examples on how the compartment API is currently used in ANA to model existing networks and protocols.

3.3 An Example

To illustrate these primitives and the notions of context and service, we describe a basic example in which two "IP-stacks" communicate over an Ethernet compartment (segment). Figure 3-1 illustrates the publish primitive inside a node M where the functional block IP-FB publishes itself in some Ethernet compartment by calling `publish(y, "*", "1.2.3.4");`

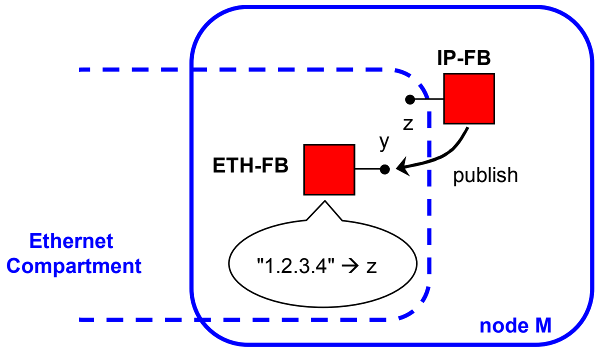

Figure 3-1. Publishing the IP service in the underlying Ethernet Compartment

The IDP 'y' identifies the compartment: it is actually bound to the Ethernet functional block (FB) inside node M. The context argument "*" is the well known context which specifies the largest possible context as interpreted by the compartment. For Ethernet, this typically means the entire segment. The service "1.2.3.4" is simply the IP address of the IP FB via which it wants to become reachable via the Ethernet compartment. Should the publish request be successful, the Ethernet FB keeps a mapping between the service "1.2.3.4" and the node-local IDP z (returned by the call to publish).

Note that this publish example is somehow similar to what happens in today's computers when one configures an Ethernet interface with an IP address via the 'ifconfig' command, and this makes the IP address resolvable via ARP.

Figure 3-2 now illustrates how an IP stack inside node N can instantiate a communication channel to the IP stack in node M by calling `resolve(e, "*", "1.2.3.4");`

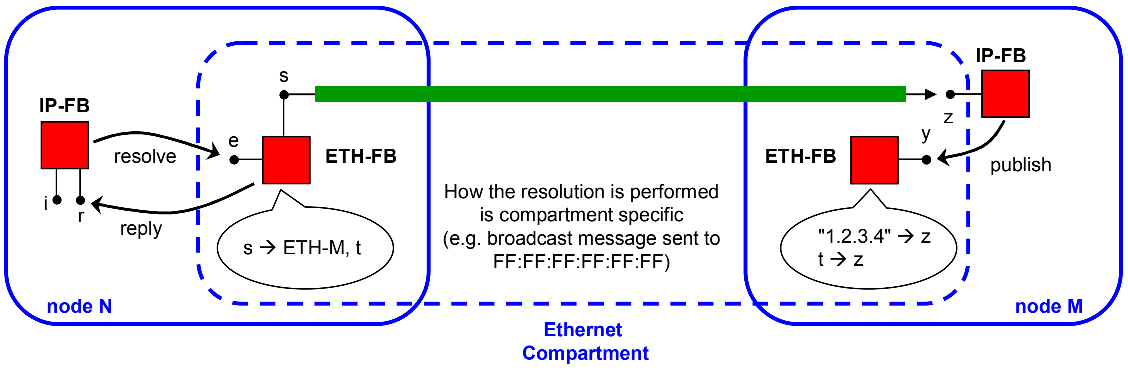

Figure 3-2. Resolving a peer node's IP service through the underlying Ethernet Compartment

The IDP 'e' identifies the Ethernet FB inside node N, the context is set to "*" to specify the largest possible context (here the entire segment), and the service is set to "1.2.3.4".

The successful request returns the IDP ‘s’ which can be used by the IP FB in node N to send data to the IP FB in node M.

In ANA, how a resolve request is handled is left to the compartment. For Ethernet, this typically results in the sending of a dedicated message containing the service being searched. Note that the context “*” translates inside the Ethernet compartment into the Ethernet broadcast address “FF:FF:FF:FF:FF:FF”.

In this example, when the Ethernet FB in node M receives the resolve request message, it finds out that the service “1.2.3.4” has been published locally and hence replies with a dedicated message containing a *token* ‘t’ which is used to demultiplex incoming packets into the IDP ‘z’ in node M. As a result, the Ethernet FB inside node N creates a local entry mapping the IDP ‘s’ into the destination MAC address ETH-M and the token ‘t’. When data is sent to the IDP ‘s’ by simply calling `send(s, data)`, the Ethernet FB in node N adds to the data an Ethernet header with destination address ETH-M and the next header 16-bit field set to ‘t’. On the receiving side, the token ‘t’ is mapped into the local IDP ‘z’ to which the payload of the Ethernet frame will be passed.

Compared to today’s network protocols, the resolution procedure of this example is the ANA equivalent of a traditional ARP resolution where an IP address is resolved into an Ethernet address.

3.4 The Lookup Primitive

So far, we have restricted our examples to the resolve primitive which, when successful, instantiates a communication channel to the destination service. However, in some cases one may not want to obtain a communication channel, or a compartment may simply not provide communication channels. This is for example the case with the DNS which resolves DNS names into IP addresses but does not create a communication channel to a name being resolved.

To support such scenarios, we introduce the lookup primitive which has the same prototype as the resolve primitive except for the return value. While the resolve primitive returns an IDP to a communication channel, the lookup primitive “simply” returns some *information*. This information may contain multiple TLV (Type-Length-Value) items such as contexts, services, and IDPs, which can potentially be used in follow-up resolve and/or lookup requests. For example, a DNS lookup request for a certain name may return a new context such as “1.2.3.4” or “2001::1” which could then be used in a further resolution request via respectively the IPv4 and IPv6 compartments.

How exactly this information should be handled is out of scope of the compartment API operation and is left to the entity performing the lookup request and getting back the information. However in the near future, we plan to provide more detailed guidelines describing how the lookup information should be handled. Our objective is that these guidelines can be used by developers in order to implement automatic procedures for handling the information returned by a lookup request.

4 FURTHER DETAILS AND OPERATIONS

4.1 The Node Compartment

The Node compartment is present in every ANA node: it is the primary hub via which all node-local interactions with ANA elements and compartments are initiated. "Communication" with the node compartment is similar to communications with any compartment, i.e. it relies on the same basic mechanisms and API described in the previous section. In a sense, the node compartment is like a "microscopic" network compartment involving just one ANA node.

4.1.1 Private views inside the node compartment

For each FB using the ANA architecture, the node compartment offers a dedicated communication workspace and a *private view* (representation) of the ANA node. Each view may be governed by its own policies, access rules, and may have strict or loose boundaries with other views. For example, a view may include shared elements provided by other applications or protocols.

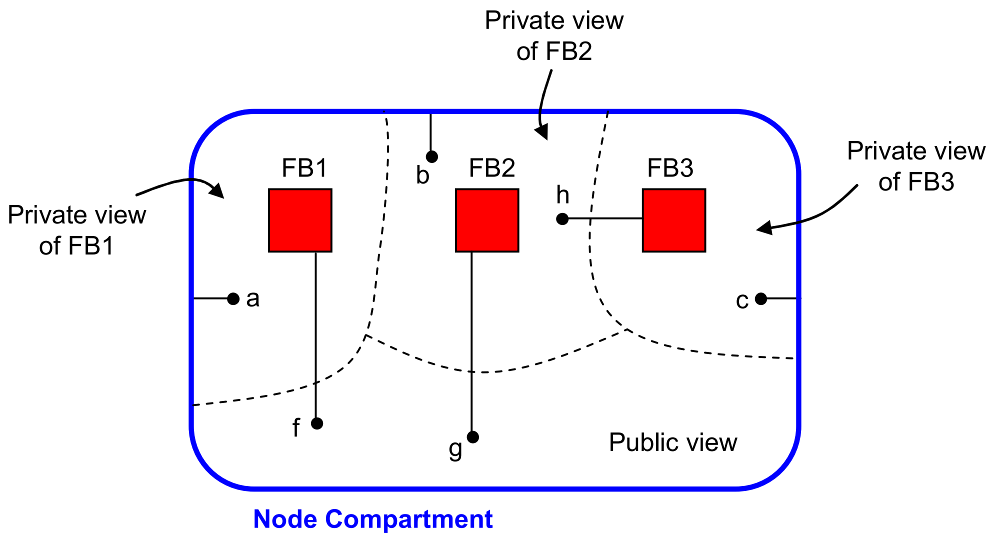

Figure 4-1. The Node Compartments and its *private* and *public views*

Upon startup, each FB receives a dedicated IDP to access the node compartment. For example in Figure 4-1, the functional blocks FB1, FB2, and FB3 respectively use the IDP values a, b, and c to communicate with the node compartment. The node compartment can hence easily control what is and what is not visible and accessible for each functional block. Note that in the figure each of these IDPs appears in the private view of the corresponding FB.

In addition to private views, the node compartment also supports a *public view*: any IDP published in this view is reachable by any other entity of the node compartment. For example in the figure, the functional blocks FB1 and FB2 are reachable via respectively IDP f and IDP g in the public view. This means they can be accessed by any other FB. In contrast, FB3 is only reachable by FB2 via the IDP h published in the private view of FB2.

Basically, the views in the node compartment allow FBs to selectively decide whether a resource (i.e. an IDP) is available to all other FBs or to only a selected FB. This simple mechanism permits the isolation of certain resources and it offers a fine-grain access control for IDPs.

4.1.2 The node compartment repository

To allow node-local FBs to publish their presence and discover other FBs, the node compartment maintains a repository that contains the list of local FBs and IDPs attached to them. In particular, this database stores the list of available (network) compartments that clients can request access to. This database also stores information about services that have been advertised and provided by other clients, and information about functional blocks that are provided by ANA, e.g., for packet encryption, data compression, transmission rate control, etc.

Typically, an FB that wants to become visible to other FBs in the node compartment publishes itself inside the node compartment with some SERVICE name. To do that, the FB uses the publish primitive of the compartment API: the IDP_c argument is set to the IDP of the node compartment, the SERVICE argument is set with the name and keywords the FB wants to appear in the node repository, and the CONTEXT is either set to "*" (i.e. for public view), "." (i.e. for the FB's own view), or to the name of the private view of another FB (how this is encoded is implementation specific).

To find out which other FBs are available inside the node compartment, an FB must use the resolve or lookup primitives of the compartment API in a similar way as the publish primitive. However when discovering other FBs, an FB can only resolve resources published in its own view and in the public view of the node compartment.

4.2 Network Compartments

A Network Compartment is a compartment that encompasses several ANA nodes and involves communication across an underlying network infrastructure.

As for any compartment, network compartments also consist of a policed set of FBs, ICs, and IDPs. In case of a network compartment, the FBs providing the compartment's communication stack are typically located on different ANA nodes. Note that ANA does not distinguish between end systems and intermediate nodes – a network compartment includes both types of node. ICs provide the communication channel between the networked ANA nodes hosting the FBs. For network compartments, ICs typically represent the underlying communication channels (e.g., an Ethernet link or a wireless channel) over which communication between the ANA nodes takes place or the abstractions of the service that is provided by an underlying compartment.

Examples of a network compartment are:

- A corporate network – whereby the hosts (i.e. PC and servers) and network nodes (i.e. routers) owned by the corporation host the FBs that provide the communication stack that is used within the corporate network compartment, and the network links (e.g., layer 2 links or VPNs) represent the ICs; the corporate network compartment also defines its own (private) addressing domain and defines the policy/trust domain.
- A wireless sensor network - whereby the sensor nodes host the FBs that provide the communication stack used within the sensor network compartment, and the wireless channels between the sensors represent the ICs.
- An overlay network – whereby the overlay nodes host the FB that provide the communication stack for the overlay, and the virtual links established between the overlay nodes represent the ICs.

4.2.1 Compartments and Layers

Network compartments can operate at the equivalent of various layers of the OSI model. That is, communication among compartment members can take place, for example, at the link layer, the network layer and/or the transport layer. A compartment can also combine the functionality of several layers (e.g., link layer and network layer) in one compartment.

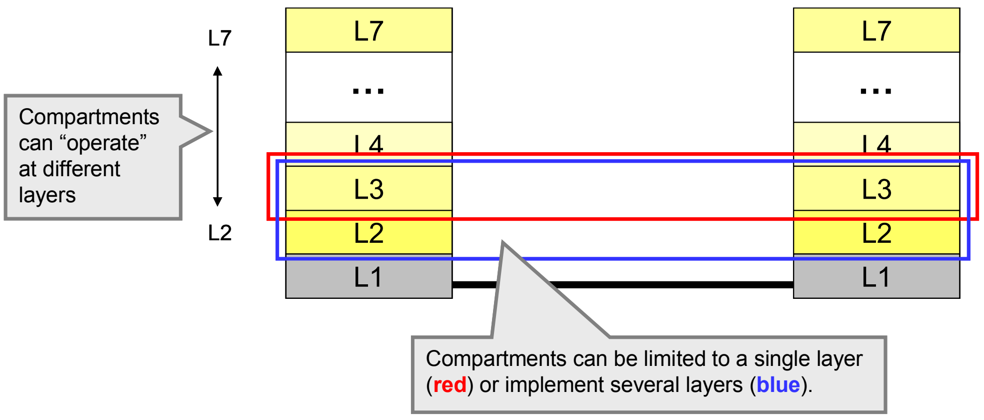
L7
L2
Compartments can "operate" at different layers
L7
...
L4
L3
L2
L1
L7
...
L4
L3
L2
L1
Compartments can be limited to a single layer (red) or implement several layers (blue).

Figure 4-2. Relation between Compartments and OSI Layers

Compartments that operate at different layers or different scopes (e.g., local vs. global), can be "layered" on top of each other. For example, a compartment that offers end-to-end communication services across heterogeneous network domains can be "overlaid" on top of the compartments providing the functionality in each of the underlying network domains.

Unlike in today's OSI implementations (e.g. in the TCP/IP stack), where layers abide a strict order (i.e. Layer 4 can only sit on top of Layer 3), compartments can be combined in more flexible ways. On the one hand, ANA enables compartments to be layered on top of any other compartments – independent of its actual semantics. In that sense, ANA is now providing the ultimate interpretation of the OSI/RM, which already envisioned such flexibility. On the other hand, as illustrated above ANA allows for a single compartment to provide semantics and functions of more than one layer (indicated by the blue rectangle), or even just some aspects of a conventional OSI layer implementation (e.g., just a control plane or management plane function). As such, compartments have much more "fluid" semantics (depending on the mission and the number of planes they span). It is even possible that a single compartment implements a whole communication stack.

Finally, compartments also extend the OSI layer model by supporting administrative policies, which enable flexible and fine-grained control of compartments entities, whereas OSI layers have no notion of administrative boundaries.

4.2.2 Layering Compartments

The ability of layering compartments on top of each other depends on the functionality (i.e. semantics) that the upper compartment expects from an "underlay" compartment and that the lower compartment can offer to an "overlay" compartment.

In order to support layering of compartment X over compartment Y, it is required that compartment X supports the particular compartment interface of compartment Y, which then simply "encapsulates" the data and opaquely passes them through the underlay compartment. In ANA this requirement is enforced since all compartments must support the API described in Section 3. This ensures that any compartment can communicate with any other compartment with a minimal and guaranteed set of interfaces.

Figure 4-3 illustrates an example whereby a FB of the overlay compartment (Compartment N) communicates with a peer FB via the underlay compartment (Compartment L1).

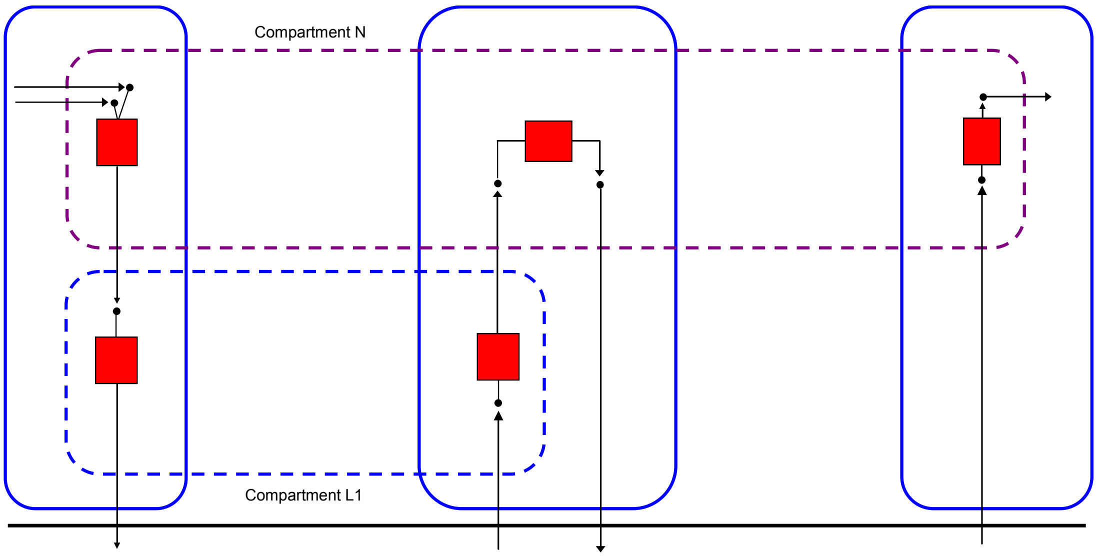

Figure 4-3. Layering of compartments – the overlay compartment makes use of services provided by the underlay compartment.

Further details on how inter-compartment communication is handled are provided in section 4.3.2.

4.3 Communication Paradigms

4.3.1 Intra-Compartment Communication

One common characteristic of ANA compartments is that all members agree on some common set of operational and policy rules. This “recipe” includes the communication principles, protocol(s) and policies to be used for intra-compartment communication. It defines for example:

- How communication elements are identified (i.e. naming and addressing).
- How identifiers are resolved (i.e. name lookup or address resolution).
- How to route information inside the compartment (i.e. routing).
- How to send/forward information (i.e. forwarding).
- What protocol to be used to communicate between peer elements.

Figure 4-4 illustrates a scenario where a set of ANA nodes (indicated in yellow) forms a network compartment in order to communicate with each other. All nodes that want to be part of the compartment must provide the compartment stack (indicated by the blue boxes) - i.e. every node has to support the necessary functionality (i.e. functional block) to join the compartment and communicate according to the compartment’s communication scheme. Nodes that provide the required functionality are able to operate according to the compartment rules, and hence can participate in the compartment (indicated by the blue circle).

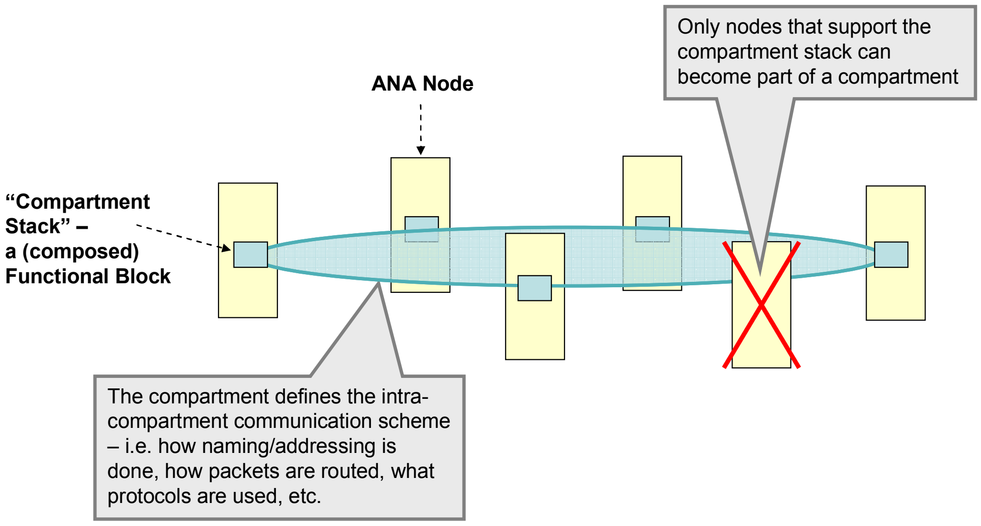
Only nodes that support the compartment stack can become part of a compartment
ANA Node
"Compartment Stack" – a (composed) Functional Block
The compartment defines the intra-compartment communication scheme – i.e. how naming/addressing is done, how packets are routed, what protocols are used, etc.

Figure 4-4. Intra-Compartment Communication

How communication inside the compartment is handled is entirely up to the compartment. Depending on the type and purpose of the compartment, different communication schemes can be applied. For example, a compartment that intends to provide the communication context for a Wireless Sensor Network, where the main goal is to aggregate sensor information on low-power devices, can implement a completely different communication scheme (i.e. addressing, routing, etc.) and protocol than a compartment that targets end-to-end communication among user terminals, such as for voice communication. While the former can use a very simple addressing and routing scheme that is able to aggregate the sensor information, the latter requires a naming or addressing scheme that supports end-to-end communication and a routing scheme that minimises end-to-end delay and jitter.

Since compartments have full autonomy on how to handle intra-compartment communication, there are no global invariants (except the compartment API) that have to be implemented by all compartments or all communication elements.

4.3.1.1 Compartment Multi-homing

ANA nodes can participate in multiple compartments. A node simply needs to support the necessary compartment functionality (i.e. the compartment stack) and join each of the compartments.

Figure 4-5 illustrates an ANA node that is multi-homed (i.e. part of several compartments at the same time). This enables the node to communicate with members of both compartments.

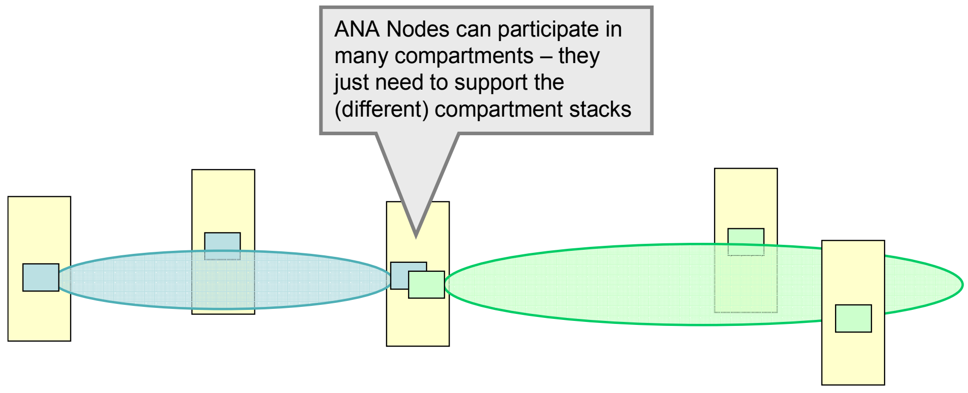
ANA Nodes can participate in many compartments – they just need to support the (different) compartment stacks

Figure 4-5. Multi-homing in several Network Compartments

It should be noted however that the fact that one node is multi-homed does not necessarily allow any of the other nodes to communicate through this node with ANA nodes in other compartments. In order to enable inter-compartment communication between ANA nodes of different compartments, the necessary interworking functionality must be provided that allows translation or proxying of communications between the different compartments.

Further details on how inter-compartment communication is handled within ANA are provided in the following section.

4.3.2 Inter-Compartment Communication

The issue of inter-compartment communication in ANA is discussed in this section. An analysis of various inter-compartment communication scenarios is followed by a set of principles or recommendations on how to best handle inter-compartment communication in ANA.

4.3.2.1 Analysis of Inter-Compartment Communication Scenarios

This section analyses different inter-compartment communication scenarios, whereby communication entities of adjacent compartments want to communicate with each other. The issues, difficulties and limitations of direct communication between peers across compartment boundaries are highlighted.

Note that in this section an identifier is simply considered some sort of an "attribute" that identifies a communication element (e.g., a FB, a network node, etc.) with regard to some semantics. Example semantics involve:

- host/interface location (e.g., an address as in IP)
- host entity (e.g., a node/host ID as in HIP)
- host role (e.g., a service identifier as in port numbers)
- host property (e.g., OS platform, performance capability)

- combined identifiers (e.g., network location and host role)

Case 1: Without a common identifier space or overlay compartment

Figure 4-6 shows a scenario whereby two communication elements of adjacent compartments want to communicate with each other. In this case, it is assumed that the two compartments operate completely independently (for example, one compartment based on IPv4 and the other based on IPv6).

Since the two compartments are completely independent, it must be assumed that there is no well-known mapping between the different communication syntax and semantics known to the communication elements of the different compartments. Only the gateway entity, which is part of both compartments, is able to communicate in both compartments.

This case further assumes that there exists no other, higher-level common identifier space (e.g., DNS) or common overlay compartment (e.g., a common transport protocol) that offers the peer entities of the different compartments a common syntax and semantics for the communication.

The following examples illustrate through real-life networking scenarios cases where such type of direct communication between peers of adjacent compartments occur.

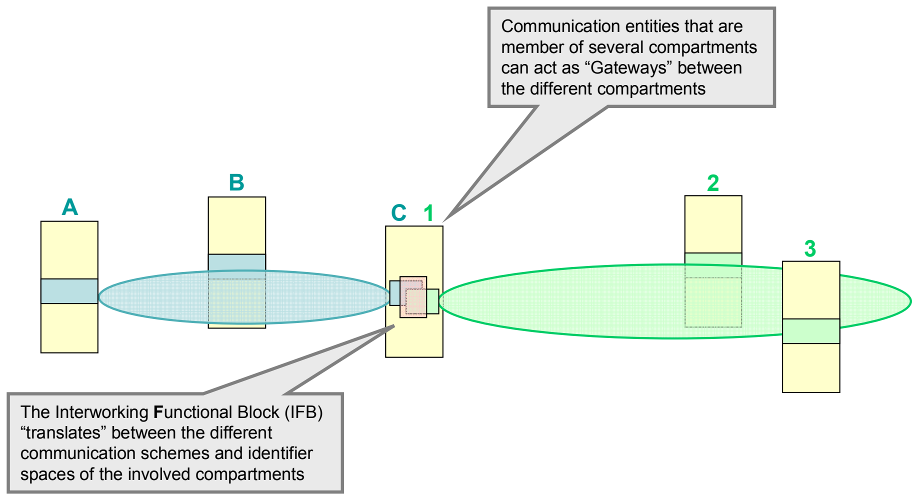
Communication entities that are member of several compartments can act as “Gateways” between the different compartments
A
B
C
1
2
3
The Interworking Functional Block (IFB) “translates” between the different communication schemes and identifier spaces of the involved compartments

Figure 4-6. Inter-compartment communication scenario without a common identifier space or overlay compartment

Example 1:

For example 1, the left (blue) and the right (green) network compartments illustrated in Figure 4-6 are envisioned to be of the same type (i.e. both use the same protocol suites) and they only use different identifier spaces (e.g., addresses or ports). Then, the gateway entity “**C/1**” only needs to do identifier translation. The required translation functionality is limited to the protocol layer within which the identifiers need to be translated. In the most typical case that will be the network layer, which makes the gateway entity “**C/1**” a

NAT box, but in theory, this translation could also occur at any other layer (e.g., the link layer) or a combination of layers (e.g., network plus transport layer).

Example 2:

For example 2, the left (blue) and the right (green) network compartments illustrated in Figure 4-6 are envisioned to be of similar type, but not the same. For example, both compartments have similar protocol suites in terms of numbers of layers and/or functionality, or provide a similar "service" to the clients. Still, translation from one protocol suite to the other (e.g., TCP/IPv4/TokenRing and TCP/IPv6/Ethernet) is doable. However, in this case the gateway entity "**C/1**" must provide more complex translation functionality than simple identifier translation. It must also be able to perform protocol translations at one or more layers. A typical example of this is a gateway that can translate IPv4 to IPv6 headers and vice versa. Note that although this function is not very difficult to implement, some loss of functionality may be encountered during the translation (e.g., IPv6 extension headers might need to be dropped if similar options do not exist in IPv4). Therefore translation is limited to support the common denominator of the features of the two protocols.

Example 3:

For example 3, the left (blue) and the right (green) network compartments illustrated in Figure 4-6 are envisioned to be of completely different nature. For example, the compartments use very different protocols, both syntactically and semantically (e.g., one provides a best effort unicast service and the other provides an episodic publish-subscribe type of functionality), and identifier semantics (e.g., one uses identifiers that encode topological information and the other uses identifiers that encode service capability). In such a case, the communication entity "**C/1**" needs to have a very complex translation capability that not only spawns across all the common denominator of the layers of both stacks, but also needs to be able to collapse/expand layer functionality from one stack to the other.

In addition to this complexity, there is also the significant challenge of naming, address and routing that needs to be resolved. It is assumed for this type of inter-compartment communication that the communication entities of the left compartment (blue) do not understand the identifier syntax and semantics of the right compartment (green). Therefore, the gateway entity "**C/1**" needs to be able to make arbitrary decisions based on some external information (e.g., policies) how to propagate data from the source entity through the one compartment and then to the destination peer in the other compartment – without common addressing and routing semantics.

<u>Conclusion:</u>

End-to-end communication across heterogeneous compartments without a common identifier space or overlay compartment is possible only if the peering compartments have a well-known translation function, both for the syntax and the semantics of the various identifier spaces and communication schemes.

Case 2: With a common overlay compartment for the communication peers

Figure 4-7 shows the same scenario, but under the condition that both communication peers are members of a common overlay compartment (e.g., a common DNS compartment or a common HIP compartment). This reduces the problem of direct communication between the peers of the adjacent compartments to the problem of routing and forwarding – as the peer entities have a common communication context.

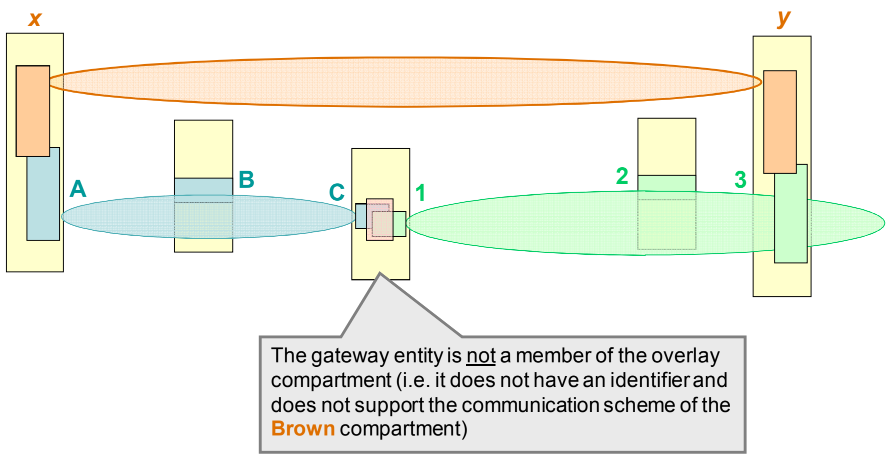
x
y
A
B
C
1
2
3
The gateway entity is not a member of the overlay compartment (i.e. it does not have an identifier and does not support the communication scheme of the Brown compartment)

Figure 4-7. Inter-compartment communication scenario whereby the communication peers are members of a common overlay compartment

This case assumes that communication entity "**A**" of the left compartment (blue) and communication element "**3**" of the right compartment (green) are also members of a common overlay compartment. The common overlay compartment gives the communicating peers ("***x***" and "***y***") a common identifier space that allows them to directly address each other, and a common communication context which allows the end hosts to understand the syntax and semantic of the exchanged data.

As the gateway element "**C/1**" is not part of the overlay compartment in this scenario, it is not able to understand the syntax and semantics of the communication at that "layer". The gateway is hence is not able to take this into account when translating between the different underlay compartments. However, most importantly, the fact that the gateway is not visible at the overlay level prevents the underlay routing mechanisms to take advantage of the routing knowledge of the overlay compartment (e.g., which gateway to choose in the underlay compartment to optimise the overlay routing path).

<u>Conclusion:</u>

As the gateway element "**C/1**" is not part of the overlay compartment, the routing knowledge that is available at the overlay compartment (i.e. how to get from "***x***" to "***y***"), cannot be used to identify the gateway node as an intermediate hop between two underlay compartments. The lack of end-to-end routing knowledge at the underlay compartment level leads to inefficient routing in case of inter-compartment communication.

Case 3: With a common overlay compartment that includes the gateway entity

Figure 4-8 shows the same scenario, but under the condition that the gateway entity "***z*/C/1**" is also visible at the overlay compartment level. As a result, when "***x***" want to send packets to "***y***", the underlay compartment can make use of the routing knowledge at the overlay level. Since the "next hop" in the overlay compartment, i.e. the gateway entity "***z*/C/1**", is also part of the left underlay compartment (blue), routing in the underlay becomes purely an intra-compartment issue. The routing knowledge of the overlay level is used by the underlay compartment to efficiently transmit the data from the source entity "**x/A**" to the correct gateway entity "***z*/C/1**".

This case also includes the scenarios where the gateway entity does not support the actual communication scheme (e.g., data plane protocols) that is used in the overlay compartment, or where the overlay compartment is only used for addressing and routing purposes. For example, a gateway entity may simply "translate" between the different underlay compartments. The classical "bridge" between two distinct LAN technologies (e.g., IEEE 802.x and IEEE 802.y) using the same address space (i.e. a common overlay compartment for addressing) is an example of such a scenario. The key difference to the Case 2 type scenarios is that the addressing and routing knowledge of the overlay compartment is used to "translate" between the different underlay compartments.

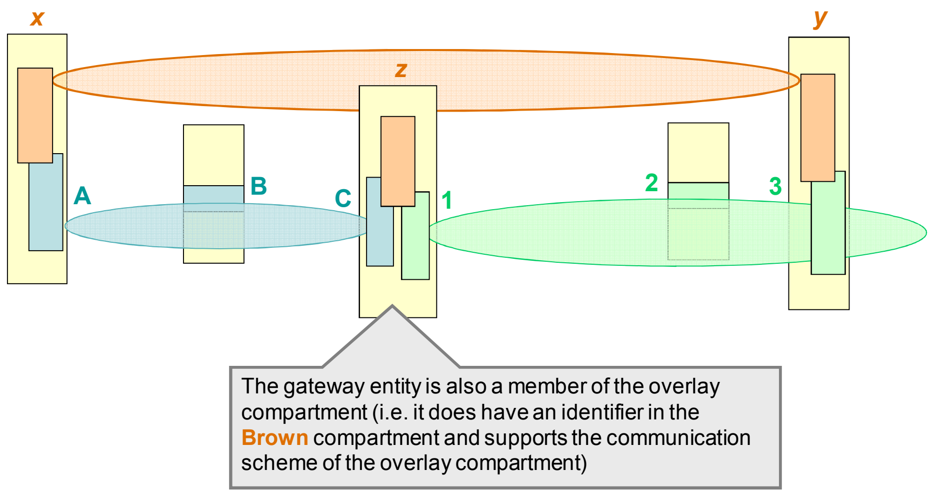
x
y
z
A
B
C
1
2
3
The gateway entity is also a member of the overlay compartment (i.e. it does have an identifier in the Brown compartment and supports the communication scheme of the overlay compartment)

Figure 4-8. Inter-compartment communication scenario whereby the communication peers and the gateway entity are members of a common overlay compartment

Conclusion:

As the gateway entity is now part of the overlay compartment, the routing knowledge of the overlay compartment (i.e. what is the "next hop" from "***x***" to "***y***") can be used to efficiently route and forward the data within and across the underlay compartments. Hence, the availability of a common overlay compartment that not only includes the communicating peer entities, but also the gateway entities (e.g., "***z*/C/1**") enables efficient inter-compartment communication across heterogeneous underlay compartments.

4.3.2.2 Principles for Inter-compartment Communication

From the above analysis of different inter-compartment communication scenarios, the following principles and recommendations can be derived:

1. *There is a need for a common compartment among communicating peers*

   Since compartments define the syntax and semantics for communication within a certain context, peers that want to communicate with each other must be part of a common compartment. Otherwise, they are not able to process (understand) the format and actual information carried in the exchanged data.

2. *It is advantageous to have a common "interworking" or overlay compartment*

   The use of a common "interworking" compartment that provides a uniform context for the communication among its members, allows hiding of potential heterogeneity at lower layers (e.g., different underlay compartments). As indicated in the conclusion above, a common overlay compartment that includes the gateway entities of the various underlay compartments also allows for efficient end-to-end routing at the underlay level.

   The fact that ANA support flexible and dynamic composition of communication stacks facilitates the introduction of such interworking "shims" as the need arises.

3. *There is a need for "interworking functionality" between adjacent compartments*

   As indicated in the analysis above, inter-compartment communication among directly adjacent or layered compartments that are of different nature require appropriate interworking functionality, which allows "translation" between the different communication syntax and semantics of the compartments. Since the required interworking functionality depends entirely on the compartments that need to interwork with each other, there is no need for a common interworking function between all compartments along the end-to-end path – i.e. different interworking functions (depending on the compartments involved) can be provided.

As illustrated in the analysis above, the focal point of the inter-compartment communication discussion are the gateway entities and the Interworking Functional Blocks (IFBs) that enable direct interworking between adjacent compartments. IFBs are conceptually part of directly peering compartments, and hence are able to "translate" between the different communication syntax and semantics of the different compartments. The IFBs are, for example, responsible for the mapping of the identifiers (i.e. name/address) as well as the translation/capsulation of the communication protocols.

Figure 4-9 shows in more detail how inter-compartment communication is handled in the case of a common "interworking" or overlay compartment. The example is based on the same scenario as the one illustrated in Figure 4-8. In this scenario, there are three entities involved. Two peer entities, namely "***x*/A**" and "***y*/3**", that want to communicate with each other despite the fact that they are not part of a common underlay compartment (e.g., because they don't share a common link layer technology), and a gateway entity "***z*/C/1**", which is part of both underlay compartments.

In order to allow for the peer entities to talk to each other, they are also part of a common overlay compartment. This offers the peer entities a common communication context, which allows the peers to address and resolve each other, and gives them a common communication protocol. Without a common communication context, the peer entities would not be able to indicate with whom they want to communicate, nor would they have a common understanding on what the exchanged messages mean.

The fact that the gateway entity is also part of the common overlay compartment facilitates end-to-end routing across the heterogeneous underlay compartments. It allows the peer entity "***x*/A**" to resolve the gateway entity "***z*/C/1**" as next-hop at the overlay level. The overlay compartment at the sender side (left) is thus able to resolve the 'next-hop identifier' at the overlay level ("*z*") through the underlay compartment.

In addition to the next-hop resolution, the sender "***x*/A**" is also responsible to handle the necessary protocol translations or the encapsulation of the payload for transmission of the information in the underlay compartment "***X***". The gateway entity "***z*/C/1**" and the receiving end "***y*/3**" perform the opposite functions, namely they decapsulate the information or performs the necessary protocol translations between the underlay and the overlay compartment.

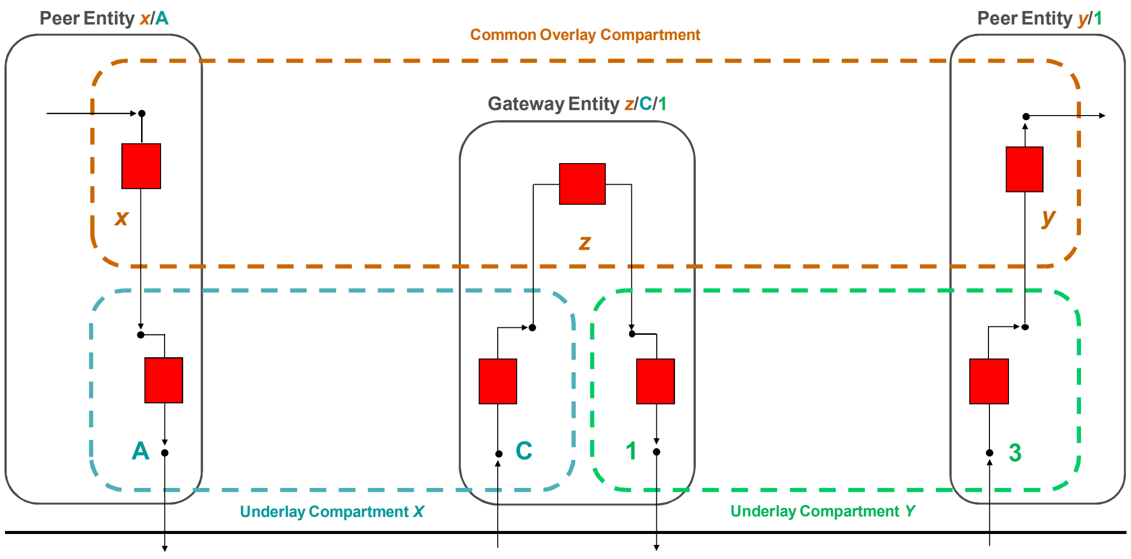

Figure 4-9. Inter-compartment communication scenario with a common overlay compartment.

5 ANA NODE

5.1 Introduction and scope

The content of this section mainly results from the work being carried out in Tasks 1.3 (Network abstractions and communication paradigms),1.4 (Communication mechanisms) and 4.1 (Testbed prototyping, management, and evaluation). While section 2 of this document provides a conceptual description of ANA and its operation, this section focuses on the description of the features that an implementation of ANA must support, and how the abstract mechanisms previously presented are provided by "the low-level machinery" of the architecture. The objective of the two sections (2 and 5) is to provide a complete picture of ANA, from the conceptual abstractions down to the mechanisms used at the implementation level.

This section is organised as follows. Subsection 5.2 provides an overview of the functionalities of the ANA node and introduces its main components. These components are then described in more detail in subsection 5.3.

5.2 Overview of the ANA Node functionalities

The ANA Node is a collection of both mandatory and optional software components that basically allow one "to run ANA" on a physical device (e.g., computer, sensor, network processor, switch, router, etc.). Three components can be identified:

- MINMEX, a minimal component for configuring and re-configuring network functionality.
- ANA playground, where optional functionality is made accessible.
- ANA hardware abstraction layer.

Each of these elements, also shown in Figure 5-1, is briefly introduced in this section.

The mandatory and underpinning component called MINMEX provides the basic low-level functionalities which are required to bootstrap and run ANA. It also provides the generic set of methods (API) that are used by "clients" of the MINMEX (i.e. protocols, applications) to interoperate with the MINMEX and the elements it hosts and with other clients of the architecture. The MINMEX is described in detail in section 5.3.

The ANA Node also implements an abstraction layer which offers a generic access to the hardware upon which it is executed. In particular, the abstraction layer has built-in functions to interface with the hosting operating system (e.g., Linux). To support legacy applications, an ANA Node should also provide an adaptation layer in environments that support e.g., Internet-style applications using the standard socket programming APIs.

This allows the re-use of existing applications in ANA without having to modify their code and re-compile them.

Besides these components, the ANA node provides a development framework and API coupled with a dedicated execution environment where the more elaborated and complex networking functionalities of ANA are placed. This flexible component, known as the “playground”, can host commodity and potentially generic networking elements such as monitoring probes, routing protocols, and cryptographical services, up to self-contained network stacks, generic peer-to-peer middleware, and complex and dedicated learning algorithms. By design, the ANA playground hosts the autonomic protocols that will exhibit a self-* behaviour.

Note that in contrast to functional blocks (FBs), information channels (ICs), and information dispatch points (IDPs) introduced in Section 2, the ANA node is not an abstraction of ANA. The ANA node is instead defined as the entity that hosts ANA elements: that is, the “node” is a property (for example “item x IS ON node y”), not an element that we need to define as an abstraction in our architecture. Only indirectly, the “ANA node property” is also captured through the compartment concept via the “node compartment” (see section 3). Via this abstraction, it is possible to query an ANA node in order to learn not only what high-level functions and services it hosts but also what network compartments it is connected to. This opens the way to offer choice to application on how to perform a communication task, while today an application *hard-codes* the network it wants to connect to (e.g., via well known protocol values such as AF_INET or AF_INET6 in socket programming).

5.3 ANA Node: architecture and components

Figure 5-1 is a conceptual representation of an ANA node whose most prominent features are the **MINMEX** area (Minimal INfrastructure for Maximal EXtensibility) and the **playground**. The MINMEX area defines the common denominator among ANA nodes, and MUST be present in all implementations. The playground hosts the optional protocols that one is free to develop with the help of the functionalities provided by the MINMEX elements.

Note that although the playground strictly contains optional components, a typical ANA implementation will be shipped with a set of “standard” features (i.e. functional blocks) to permit access to basic network compartments.

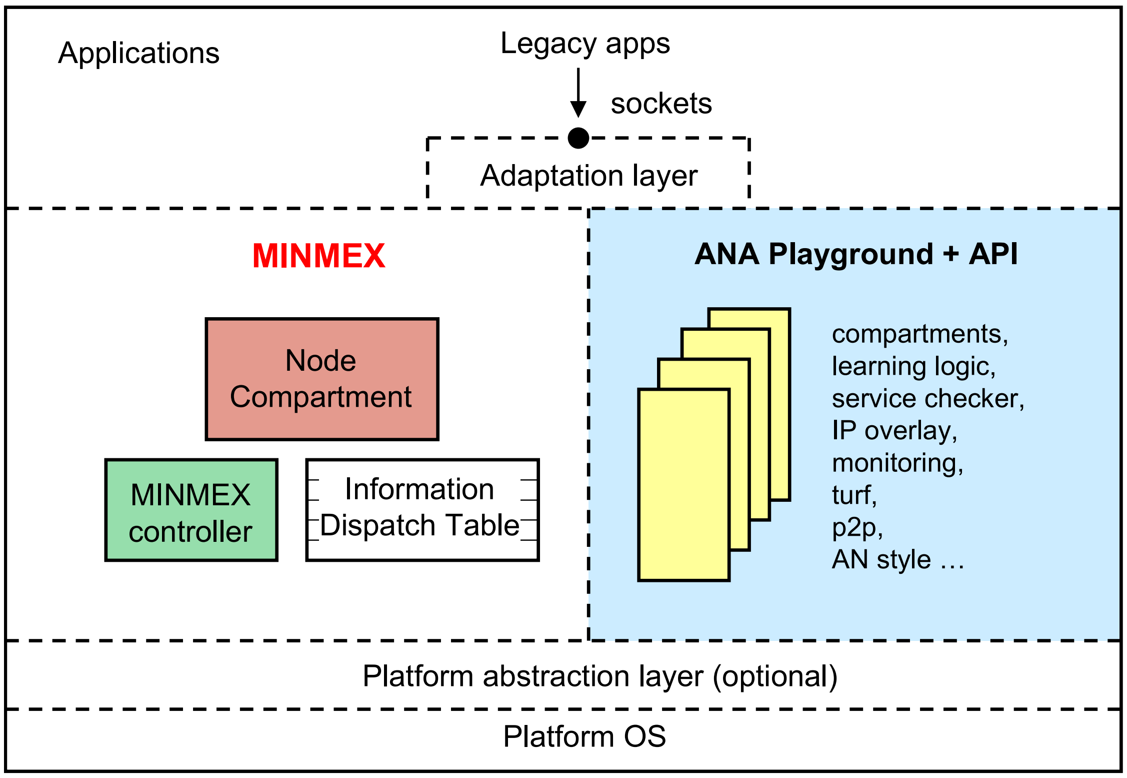

Figure 5-1. The components of an ANA node, embedded in an OS context.

The "MINMEX approach" employed in the design of ANA permits us to focus on a simplified core and declare richer and more complex functionality as *extensions* of the architecture. In a sense, MINMEX forms the constant part of an ever evolving network: it is more an architecture framework than a network architecture by itself. Thus, a central role of MINMEX is to provide the generic functionalities to instantiate and run network compartments code and to provide a dedicated execution environment for each "instance" of a network compartment.

The operations offered by the MINMEX can be characterized as follows:

- Means to forward packets between applications, functional blocks, and interfaces.
- The Node Compartment, allowing access to ANA communication mechanisms, protocols, and compartments.
- A MINMEX controller which performs a continuous assessment of the basic operation of an ANA node.

In the following three subsections, we provide a more detailed description of the components of the MINMEX that have not been described yet (the Node Compartment was already described in Section 4.1).

5.3.1 The Information Dispatch Framework (IDF)

As introduced in section 2 of this document, the distribution of data packets in ANA is achieved via IDPs (Information Dispatch Points). To recap, data packets are always sent to an IDP which is itself bound to a functional block (FB) or an information channel (IC) (at least conceptually). This approach provides ANA with powerful indirection capabilities that permit seamless redirection of packet flows without the sender not (necessarily) being aware of any change: it will still send data to the same (starting) IDP.

As shown by the set of examples in section 3, data following a communication path is typically sent to a starting IDP and then passed on through a chain of functional blocks to the downstream IDPs. This process is somehow similar to the hop-by-hop forwarding of data in today's Internet. However while in the Internet a "hop" in the forwarding process is typically a network hop (i.e. data is passed on to a neighbouring network node), ANA extends and generalizes this concept to data being forwarded inside a node among FBs and to other (network-wide) nodes via ICs. Note that while (strictly speaking) data is sent to an IDP, it is up to the entity bound to the IDP to know to which "next hop IDP(s)" the data should be forwarded. Hence while conceptually IDPs are powerful abstractions of ANA, implementation-wise an IDP is actually an entry in a functionally enriched forwarding information base (FIB).

The forwarding procedure inside ANA is illustrated in Figure 5-2, which shows data being 1) received by a node compartment, 2) forwarded inside the node compartment and 3) forwarded to a distant node via an information channel. Data is received by IDP *b* which is bound to FB *f2* that has next-hop IDP *c*. Data dispatching carries on until reaching IDP *e* which calls a low-level function provided by the ANA Node abstraction layer that sends data via a network interface. Typically for link-layer forwarding, IDP *e* will store the information (e.g., MAC address) needed to reach the next hop network node. Note that the figure also provides an abstract representation of the forwarding table being used.

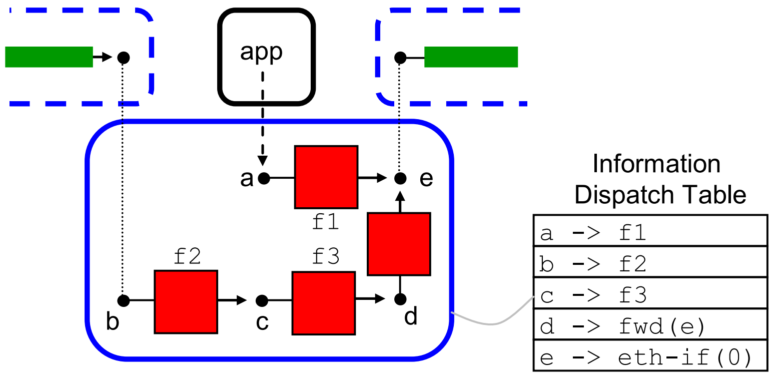

Figure 5-2. Data dispatch inside ANA.

In practice and as described in detail in [9], the interaction with the information dispatch framework is achieved via dedicated APIs: at the conceptual level one manipulates IDPs and has no direct access to the (real) internal data structure. With these APIs the (local) instance of a network compartment can create and delete IDPs and set per-IDP mandatory information such as the function bound to the IDP.

A critical feature of the information dispatch framework (IDF) is that it is fully address- and name-agnostic: IDPs are identified by local labels (i.e. numbers) which have no networking meaning whatsoever. This is a key requirement since the IDF must dispatch data that (conceptually) belong to radically different compartments with any sorts of addressing and naming schemes. This *neutral* position allows this core component to support any new addressing or naming scheme that can be developed in the future.

To separate concerns, IDPs are stored in Information Dispatch Tables (IDTs). An IDT typically contains all of the (node-local) IDPs that belong to some communication context, e.g., a given compartment or a given view of the node compartment. This conceptual separation prevents, e.g., unauthorized manipulations of IDPs or excessive resource (e.g., memory) consumption by misbehaving or faulty components. Note that in terms of implementation there may only be one table where each entry (IDP) would contain a field indicating which IDT it belongs to.

In conclusion, one should note that while the information dispatch framework provides flexible and powerful mechanisms to create and dynamically modify data paths, it is not responsible for setting up and changing these paths. That is, the “control” is typically left to higher level entities such as the functional composition framework, ANA applications, or individual functional blocks that can configure and modify “data paths” according to some flow specifications and to the state of the network.

5.3.2 The MINMEX Controller (MC)

The MINMEX controller (MC) performs a continuous assessment of the basic operation of an ANA node. It is truly autonomous and basically performs a sanity and health check of the components running inside the node. The core objective of the MINMEX controller is basically to “protect” the MINMEX elements from faulty or misbehaving components in order to guarantee the performance of the ANA subsystem. Keeping such control separate and not embedded in all the other elements is beneficial for two main reasons. First, it simplifies the design and code of the other MINMEX elements and second, it allows control operations to be centralised and coordinated in one entity. The main drawback is that all control relies on a single entity which becomes a very sensitive component of the architecture. However, the MINMEX controller could potentially be implemented in a distributed way which would greatly reduce the risks of a total unrecoverable failure.

The rest of this section outlines some of the fundamental activities of the MINMEX controller (MC). This description is not exhaustive and additional tasks may be added in the future as needs appear. However, note that the following control operations are mandatory and must be supported by all implementations of ANA.

- The MC periodically controls all the forwarding paths and states of the information dispatch tables. In particular, it must detect forwarding loops which may have been accidentally created. If a loop is detected, the MC must prevent entities to send data on the loop: this interfering action must generate an appropriate error code back at the sender side. Data circulating inside a loop must be destroyed.
- The MC performs a garbage collection of unused or expired IDPs that may have become orphan IDPs if the entities using them are malfunctioning or have crashed. Such IDPs must be removed from the system in order to free resources. Note that this verification may become redundant or obsolete if IDPs are stored in a soft state manner.
- In a similar manner, the MC performs a garbage collection of the entries in the node compartment. This task may also become redundant or obsolete if the entire repository in the node compartment relies on soft state storage.
- In addition, the MC periodically controls the state of all the active functional blocks running in the system. If the MC detects that a functional block is malfunctioning or crashed (via e.g., the failure of a polling mechanism), the MC must either remove the FB from the system or try to re-instantiate this functionality. The action to be taken in the case of a failure may be specified by the FB itself upon start-up.
- The MC provides the lowest level of access rights and control for accessing IDPs. For example and as already described, the MINMEX may maintain separate IDTs for different communication contexts: in that case, the MC must prevent un-authorised access to IDTs and return an appropriate error code to the entity performing the rejected operation.

5.3.3 The playground

The playground contains all the optional extensions of the base ANA node. This is where complex protocols and functions are implemented and where network-wide autonomicity is achieved. It is the area where variety will exist, perhaps even programmability in an active networking sense.

Typically, the playground will host both commodity and dedicated functionalities. Commodity elements are “public” components that one can re-use in order for example to compose more complex functions in an on-demand manner. This includes, e.g., cryptographical primitives, compression schemes, queuing systems, reliable transmission schemes, generic learning algorithms, and error recovery codes. How such elements can be dynamically composed is described in deliverable D2.12.

The playground may also contain complex components that are specialised for a certain task and can hardly be re-used outside their initial design space. This category includes, e.g., fully-contained network stacks/heaps (”compartments code”), highly dedicated and optimized protocols such as delay-tolerant transmission schemes, and schemes for network coding. This classification is highly subjective and does not exist inside the playground: we here merely want to highlight that functions inside the Playground range from simple re-usable components to complex and highly specialized elements.

Note that in practice, the protocols of the playground interact with the MINMEX elements via a specific API that allow them to, e.g., initially communicate with the node compartment and then with network compartments, instantiate functional blocks and information dispatch points, and create information channels to distant peers. The current version of the API is described in details in the documentation of the ANA Core software as part of Deliverable D1.11 [9].

6 CONCLUSIONS

As stated previously in this document, the main intention of the Blueprint is to describe the reference model of the ANA network architecture. This includes a description of the basic abstractions and communication paradigms. Another aim of this document is to guide the prototyping of the ANA communication system throughout the project.

At the coarsest level, the core abstraction of ANA is the "Compartment" which encompasses the notions of networks and realms and is the basic unit for the interaction and federation of networks. A key feature is that compartments reduce complexity by hiding the compartment internal communication complexity to the "outside world", which interfaces with a given compartment via a generic API that all compartments must support. With compartments, networks can be decomposed into smaller and easier manageable units: each compartment has indeed full autonomy on how to handle internal communication (i.e. naming, addressing, routing, etc.). The boundaries of a compartment are typically based on technological and/or administrative boundaries or policy domains. ANA anticipates that many compartments co-exist and will provide the necessary functionalities such that compartments are able to interwork on various levels.

At a more detailed level, compartments contain three kinds of abstraction: Functional Blocks (FBs), Information Channels (ICs), and Information Dispatch Points (IDPs). These abstractions are used to model the "internals" of a compartment and the service the compartment offers. The degree of details at which a compartment exposes its internal abstractions is specific to each compartment and ranges from full visibility to total opaqueness.

At the lowest level, the underpinning component is the ANA Node, which is the entity that hosts the ANA elements and functionalities. It is the main interface to the host operating system or underlying hardware, and also provides an adaptation layer allowing legacy applications to be reused in an ANA context. The core and mandatory part of the ANA Node is the MINMEX framework, which provides the basic low-level functionalities required to bootstrap and run an ANA node. Beside the MINMEX framework, the ANA node provides a development framework – the "playground" – where the more elaborated and complex networking functionalities of ANA are placed. For example, compartment-specific modules are executed in the Playground.

The feedback obtained during the first and second prototyping phases (M13-M24 and M30-M36) has been used to extend and refine the first versions of the Blueprint. Further feedback that will be gained during the remaining prototyping phase will eventually lead to the specification of the final version of the Blueprint in December 2009.

7 REFERENCES

[1] H. Zimmermann, OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. IEEE Transactions on Communications **28**(4), April 1980, pages 425-432.

[2] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, The Click Modular Router. ACM Transactions on Computer Systems **18**(3), August 2000, pages 263-297.

[3] C. Tschudin and R. Gold, Network Pointers. In Proc. of the first ACM Workshop on Hot Topics in Networks (HotNets-I), October 2002, Princeton, USA.

[4] I. Stoica, D. Atkins, S. Zhuang, S. Shenker, and S. Surana, Internet Indirection Infrastructure. In Proc. of ACM Sigcomm'02, April 2002, Pittsburg, USA.

[5] D. Clark, J. Wroclawski, K. R. Sollins and R. Braden. Tussle in Cyber-space: Defining Tomorrow's Internet. In Proc. of ACM SIGCOMM, Pittsburgh, PA, USA, August 19-23, 2002, pp. 347-356.

[6] D. Clark, R. Braden, A. Falk and V. Pingali. FARA: Reorganizing the Addressing Architecture. In Proc. of ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA), Karlsruhe, Germany, Au-gust 2003, pp. 313-321.

[7] J. Pujol, S. Schmid, L. Eggert, M. Brunner and J. Quittek. Scalability Analysis of the TurfNet Naming and Routing Architecture. In Proc. of 1st ACM Workshop on Dynamic Interconnection of Networks (DIN 2005), Cologne, Germany, September 2, 2005, pp. 28-32.

[8] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield, Plutarch: an Argument for Network Pluralism. In Proc. of the ACM Workshop on Future Directions in Network Architecture, August 2003, Karlsruhe, Germany.

[9] G. Bouabene (Editor), D1.11: Second Public Stable Release of the ANA Core Software. Workpackage 1, Deliverable D1.11, December 2008.

APPENDIX

In this appendix, we provide examples of how existing networking technologies and protocols can be modelled using ANA. The objective is to illustrate the use of the CONTEXT and SERVICE arguments and highlight the flexibility of these constructs with concrete examples.

Ethernet Compartment

In this compartment the CONTEXT defines the scope on the Ethernet segment. The “*” CONTEXT means the entire segment (FF:FF:FF:FF:FF:FF) while the “.” CONTEXT means the FB-local scope (in Linux the loopback interface has MAC address 00:00:00:00:00:00).

For Ethernet, publishing a SERVICE in the “*” CONTEXT means this service will be resolvable by anyone on the entire segment. Here is an example, with the IDP e being attached to some Ethernet FB in node A, and IDP f being attached to some Ethernet FB in node B.

```
In node A, some IP FB does:  publish(e, "*", "1.2.3.4");
In node B, some IP FB does:  resolve(f, "*", "1.2.3.4");
```

and the IP FB in node B obtains an information channel (IC) to the IP FB (i.e. the SERVICE "1.2.3.4") in node A.

Note that because a resolution inside the Ethernet compartment is basically the equivalent of an ARP request, most publish and resolve use the “*” CONTEXT. Note that similar to ARP, there is a problem if two FBs publish the same SERVICE in the “*” CONTEXT because a later resolution will receive 2 different answers (ARP suffers from the same problem).

However, if one wants to resolve a particular service at a known location (11:22:33:44:55:66), it is possible to do

```
resolve(e, "11:22:33:44:55:66", "whatever_service");
```

in order to obtain an information to some service at a particular location (i.e. given as the CONTEXT of the resolution request).

Note that with the Ethernet compartment publishing something at a remote location makes no sense so doing the following publish fails if the FB making the API call is not located on 11:22:33:44:55:66.

```
publish(e, "11:22:33:44:55:66", "my_service");
```

IP Compartment

For this compartment, the "*" CONTEXT means either the subnet broadcast address (e.g. 10.1.2.255 for the subnet 10.1.2.0/24) or the undefined broadcast 255.255.255.255. In any case, such broadcast is restricted to the link segment. The "." CONTEXT means the loopback address 127.0.0.1 (node-local scope).

Publishing a service in the "*" CONTEXT means that this service is resolvable from anywhere. However, doing a resolve in the "*" CONTEXT means resolution is restricted to the link segment, because flooding the entire IP compartment is not feasible (makes no sense). Typically, one has to better specify the CONTEXT. Here is an example with IDP i being attached to some IP FB in node A and IDP j being attached to some IP FB in node B.

```
 In node A, some web server FB does:  publish(i, "*", "tcp:80");
 --> let say the IP FB in node A has IP address 1.2.3.4

 In node B, some FB does:  resolve(j, "1.2.3.4", "tcp:80");
 --> to obtain an IC to the web server in node A
```

Note that the SERVICE "tcp:80" is of course different based on the CONTEXT because in IP the CONTEXT defines the location in the IP compartment.

For multicast, one could publish a given service with the corresponding IP multicast address (resolution is equivalent). For example:

```
 In node A, publish(i, "224.0.0.9", "rip_routing");
 In node B, publish(j, "224.0.0.9", "rip_routing");
 In node C, resolve(k, "224.0.0.9", "rip_routing");

 --> to obtain at node C a broadcast IC to all instances (i.e. in nodes
A and B) of RIP on the link segment.
```

DNS Compartment

The DNS compartment was not implemented in ANA, but we provide below a possible design for it. In this example, the CONTEXT in DNS is the record type: that is it defines the resolution scope inside the range of possible DNS records.

Here are some examples, with IDP d being attached to some DNS FB. Also note that the resolve primitive makes no sense in the DNS compartment because this compartment does not offer information channels, it only returns some information so we have to use the lookup primitive.

To perform a lookup in the DNS compartment, one can do:

```
 lookup(d, "A", "www.dummy.com");
 --> returns an IPv4 address of www.dummy.com
 lookup(d, "AAAA", "www.dummy.com");
 --> returns an IPv6 address of www.dummy.com
 lookup(d, "MX", "dummy.com");
 --> returns the MX (email server) record for dummy.com
```

To publish en entry in the DNS compartment, one issue is that one has to pass 3 arguments: the CONTEXT ("A" or "AAAA" or "MX" ...), the DNS name **and** the IP address. This means the SERVICE argument must be used to encode 2 parameters (i.e. the DNS name and the IP address), so we have to do some formatting trick to fit 2 arguments in the SERVICE.

```
publish(d, "A", "www.example.com|1.2.3.4");
--> note that the SERVICE is used to pass 2 arguments
```

This is not very elegant, but in the mean time there is no equivalent in the Internet/socket world of today: that is, today an application cannot publish an entry in the DNS. Another way of doing this would be that the DNS compartment FB could check the CONTEXT of the FB making the publish call and use it to perform the publish. For example, let assume that an IP FB with IDP i attached to it has the (source) CONTEXT=”1.2.3.4”. The IP FB would do:

```
publish(d, "A", "www.example.com");
(IDP i is implicitly passed to the DNS FB)
```

The DNS FB could check the CONTEXT of IDP i and obtain the value “1.2.3.4”. Hence with this technique there is no need to pass 2 arguments via SERVICE. However, this requires that any FB using the DNS compartment must set its local CONTEXT value so that it can be obtained by the DNS compartment FB.

Publish/Subscribe protocol

Publish/subscribe protocols are quite particular because with this communication model some content can be “pushed” in the network even though there is no active receiver. A receiver can then later notify its interest for the content which is then “pulled” to the receiver.

In ANA, this means that a publish/subscribe compartment must accept resolve and send requests even if there is no reachable receiver. The content must be stored in the compartment so that it can later be retrieved when a receiver notifies its interest via the publish primitive.

Here is an example, with the IDP a being attached to some pub/sub FB in node A, and IDP b being attached to some pub/sub FB in node B.

In Node A, an FB providing some content does:

```
resolve(a, "*", "some_content"); (this returns IDP s)
send(s, some_content); (this inserts the content in the compartment)
```

In Node B, an FB willing to get that content can later do:

```
publish(b, "*", "some_content");
```

to notify its interest for the content and retrieve it immediately (remember that in ANA the publish primitive sets up a callback function for receiving data).