Deliverable D.2.10

Implementation of Inter-Compartment Communication Schemes

Implementation Report
## Abstract:

This document introduces the protocols and implementations of the two inter-compartment communication schemes that have currently been developed in ANA. The core parts of this deliverable are actually the two software implementations: the goal of this companion document is to briefly summarize the design and status of the implementations that have been included in the development branch of the ANA software.

## Keywords:

ANA, inter-compartment, routing.
# Table of content

1 Introduction ................................................. 4

2 The Compartment Finder (cfinder) .......................... 5
   2.1 Overview of operation .................................. 5
   2.2 Technical details ......................................... 6
      2.2.1 Neighborhood discovery ............................ 6
      2.2.2 Interacting with the cfinder compartment .......... 8
      2.2.3 Routing inside the cfinder compartment ............ 8
      2.2.4 A proxy service for inter-compartment information channels 9
   2.3 Conclusion .................................................. 11

3 Overlay-based Inter-Compartment Communication (ANAturf) 11
   3.1 Terminology ................................................. 12
   3.2 Specific design aspects ................................... 12
   3.3 Communication flow example ............................... 13
   3.4 ANA-related details and usage examples .................. 14
      3.4.1 Overview ................................................. 14
      3.4.2 Startup .................................................. 15
      3.4.3 Client-side interface ................................. 16
   3.5 Conclusion .................................................. 16

4 Summary ...................................................... 16

References ..................................................... 17
1 Introduction

The ANA (Autonomic Network Architecture) project aims at providing a framework to flexibly host, interconnect, and federate multiple heterogeneous networks in an autonomic way, i.e. without requiring active human intervention. For achieving this goal, ANA introduces at its macroscopic level an abstraction used to wrap all possible (and future) network instances in a generic container: the network compartment. Here the verb “wrap” and adjective “generic” actually refer to a minimal API that must be supported by all compartments and permits to “talk to” all network compartments in a generic manner without requiring any a priori knowledge of how the compartment internally operates.

In ANA, each network compartment is free to internally run its own set of networking protocols and algorithms for addressing, naming, and routing. The only requirement from ANA is that each compartment must support the generic API to allow all possible (and unforeseen) interactions with any other component of the architecture. In other words, each compartment operates as a black box which supports a generic and flexible interface with the “outside world”. The motivation for supporting heterogeneous network instances is that we envisage (as others [1, 2]) that future inter-networks will consist of a rich and diverse collection of networks with multiple and potentially incompatible namespaces.

The main design challenge in such a catenet\(^1\) is to offer a global network connectivity over a heterogeneous environment without having to design dedicated translation mechanisms between all possible compartment-pairs. Since the “design philosophy” of ANA is to encourage variability at all levels of the architecture, we have proposed in [3] two different but complementary ways for performing inter-compartment communication and routing. In 2008 and as the core contribution of Deliverable D.2.10, these two schemes have been implemented. As a companion document of D.2.10 (the core part of D.2.10 is the software), this report summarizes the design of these two implementations and their integration into the ANA software.

In order to have a self-contained document, sections 2 and 3 start with a brief summary of each the two approaches that have been developed. They then provide some technical details about the two implementations. Section 2 considers the “Compartment Finder” scheme developed by UBasel, and Section 3 examines the “TurfNet” protocol developed by NEC. Finally, Section 4 concludes the document.

\(^1\)“concatenation of networks”, i.e. original concept of inter-network.
2 The Compartment Finder (cfinder)

The Compartment Finder (cfinder) framework was already described in the ANA Deliverable D.2.3 [3]. However, in order to provide a self-contained implementation report, we briefly summarize in the next subsection its main features and operation. After this summary, we provide some implementation details in subsection 2.2 in order to document the main aspects and features of the current implementation.

2.1 Overview of operation

To provide inter-compartment routing, the cfinder compartment creates an overlay structure on top of existing underlying compartments which have a limited scope and reachability. Compared to classical overlay approaches for global routing (e.g. the Internet) where each entity in the federated underlays must have a global identifier in the overlay, the compartment finder only requires that the namespace of each of the underlying overlays has an identifier in the global overlay. In other words, an entire underlying network is mapped in the global overlay with just one unique name.

To do that, we use the string representations of the names/addresses/identifiers of each underlying compartment and create a regular expression for matching this namespace. The regular expressions are then exchanged in the cfinder overlay in order to create inter-compartment routing entries based on regular expressions, with each entry mapping an entire destination network. This is illustrated by Figure 1.

![Figure 1: Overview of the cfinder inter-compartment routing.](image-url)
In this figure, the node ID1 in the cfinder routing overlay has a routing entry for the regular expression $(\{x\{1,4\}:\}^7)x\{1,4\}$ with next hop node ID2. This regular expression actually matches any IPv6 address\(^2\), so in essence this defines an inter-compartment route to the IPv6 compartment. To communicate with one another, the two nodes belonging to the cfinder compartment use the underlying IPv4 compartment, as shown by the two information channels linking them.

Based on this inter-compartment routing entry, a node in the IPv4 compartment can query a node of the cfinder compartment in order to obtain an information channel to a remote IPv6 destination. Hence while the entities of the cfinder compartment use other compartments to communicate, they serve these compartments by offering them inter-compartment communication services.

### 2.2 Technical details

In this subsection, we provide some technical details for the implementation of the cfinder FB. We first focus on the general aspects of the FB and how it interacts with other FBs via the compartment API. We then introduce the routing scheme used internally in the cfinder compartment, and we finally provide some details on a specific proxy service that had to be developed to support inter-compartment communications.

#### 2.2.1 Neighborhood discovery

As already stated in the previous section, the cfinder functional blocks (FBs) use other compartments to create information channels (“peerings”) with remote peers and exchange inter-compartment routing information in the form of regular expressions. To create such peerings upon startup, each instance of the cfinder FB receives some configuration options providing information to reach other (remote) cfinder peers. This peering information may contain multiple tuples in the form `<Compartment name, CONTEXT, SERVICE>`, which each tuple being used by the cfinder FB to resolve() an information channel to a remote peer.

For example and based on the simple scenario of Figure 1, the configuration information of Node B would contain the tuple `<"ipv4", "9.8.7.6", "cfinder">` which would lead to the following primitive calls:

```r
i <- resolve(NODE_LABEL, "+", "ipv4");
```

to query the node compartment with the keyword "ipv4" and obtain the IDP 'i' to reach the IPv4 compartment, followed by the primitive call

```r
s <- resolve(i, "9.8.7.6", "cfinder");
```

\(^2\)For simplicity, this regular expression does not match IPv6 addresses that contain the :: notation for consecutive zeros.
to obtain the information channel $s \rightarrow t$ to reach the $\text{cfinder}$ FB in Node C (to simplify the figure, this FB is not shown). The initial configuration can of course contain multiple tuples so that a $\text{cfinder}$ FB can setup multiple information channels with remote peers.

In addition to this initial configuration, each $\text{cfinder}$ FB also tries to discover remote peers via all the network compartments available locally on the ANA node. To do so, it issues the primitive call

$$\text{IDP list} \leftarrow \text{lookup}(\text{NODE_LABEL}, "\star", "\text{compartment}");$$

in order to obtain a list of IDPs to reach local compartment FBs. Then for each IDP 'k' in the list, the $\text{cfinder}$ FB issues a primitive call

$$x \leftarrow \text{resolve}(k, "\star", "\text{cfinder}");$$

which, if there exists any $\text{cfinder}$ service reachable in the "\star" CONTEXT, returns an information channel to a remote $\text{cfinder}$ peer in the "\star" CONTEXT of the compartment. This is for example useful to discover peers on an Ethernet segment or inside an IP subnet without knowing their exact CONTEXT (i.e. here an address) value.

Upon startup and to make itself visible to local FBs and remote peers, the $\text{cfinder}$ FB inside an ANA node publishes itself in the Node Compartment and in all the compartments available locally. This is achieved with three primitive calls:

$$c \leftarrow \text{publish}(\text{NODE_LABEL}, "\star", "\text{cfinder+compartment}");$$

which returns the IDP 'c' via which the $\text{cfinder}$ FB is reachable in the Node Compartment. As before, the FB then obtains the list of compartments available locally:

$$\text{IDP list} \leftarrow \text{lookup}(\text{NODE_LABEL}, "\star", "\text{compartment}");$$

and then issues, for each IDP 'k' in the IDP list obtained, the primitive call

$$h \leftarrow \text{publish}(k, "\star", "\text{cfinder}");$$

to make itself visible in the "\star" CONTEXT of all the local compartments.

With these simple API calls, a $\text{cfinder}$ FB can setup information channels with remote peers and start exchanging routing information as described in subsection 2.2.3. However before being able to announce any inter-compartment routing information, a $\text{cfinder}$ FB must first receive such information from local compartments as now described.
2.2.2 Interacting with the cfinder compartment

When a compartment FB wants to be visible and reachable in the cfinder compartment, it must publish() in the cfinder compartment a regular expression matching its namespace. For example, an IPv6 FB would achieve this by calling the following primitives:

\[
\begin{align*}
    c & \leftarrow \text{resolve(NODE\_LABEL, "\*", "cfinder")}; \\
    i & \leftarrow \text{publish(c, "\*", "(x{1,4}:){7\}x{1,4}"});
\end{align*}
\]

to query the node compartment with the keyword "cfinder" and obtain the IDP 'c' to reach the cfinder compartment FB. This is followed by the primitive call

\[
\begin{align*}
    i \leftarrow \text{publish(c, "\*", "(x{1,4}:){7\}x{1,4}"});
\end{align*}
\]

which returns the IDP 'i' via which the IPv6 FB is now reachable in the cfinder compartment.

If another FB now wants to obtain an information channel to a compartment FB handling a certain namespace, it must query the cfinder compartment with the target name. Assuming the FB already obtained the IDP 'c' of the cfinder compartment FB, it would issue the following primitive request to obtain an information channel to some IPv6 FB:

\[
\begin{align*}
    j & \leftarrow \text{resolve(c, "\*", "2001:620:200:1::1");}
\end{align*}
\]

which returns the IDP 'j' via which some (local or remote) IPv6 FB is reachable in the cfinder compartment (wrt. to the previous example, the IDP 'i' would be returned). Note that the content of the name (here "2001:620:200:1::1" does not matter as long as it matches the target namespace; however, an FB would typically use the name with which it then wants to make a further resolve() request as shown below. To then obtain an information channel to some remote service in the IPv6 compartment, the FB would issue a primitive call like:

\[
\begin{align*}
    s & \leftarrow \text{resolve(j, "2001:620:200:1::1", "tcp:80"});
\end{align*}
\]

which returns the IDP 's' via which some service (here "tcp:80") is reachable in the IPv6 compartment in the context "2001:620:200:1::1".

2.2.3 Routing inside the cfinder compartment

The routing inside the cfinder compartment is based on the regular expressions published in cfinder FBs. Like with any other routing protocol, the peers exchange regular expressions along with a cost value that permit to decide whether which out of two similar entries is best to keep. In our current implementation, we use the hop distance (inside the cfinder overlay) as the cost of regular expression entries.

The routing algorithm itself is nothing new. To avoid spending too much time re-implementing a complex routing protocol algorithm and rather focus on the key
aspects of the \texttt{cfinder} scheme, we have implemented a simple routing algorithm similar to the Pulse protocol [4] for MANETs. Basically, each protocol instance periodically floods its routing entries to all its neighbors, and each neighbor decides to further re-flood a routing entry if and only if it decides to use this entry in its routing table. With this simple scheme, the algorithm essentially creates spanning trees rooted at the peers whose entries are being used by neighboring nodes.

While this routing algorithm is very simple and robust, it does not really scale to large networks. However, the implementation is modular enough so that it is easy to replace it in the future with a better and more complex algorithm. Nevertheless, our research focus is on the \texttt{cfinder} framework and not on routing scalability, so this simple routing algorithm is good enough for testing our framework.

A more serious implementation issue relates to the way ANA handles API requests. Since the \texttt{cfinder} compartment basically offers a virtual access to remote network compartments, we had to develop a proxy service in order to encapsulate API requests via the \texttt{cfinder} compartment.

\subsection{2.2.4 A proxy service for inter-compartment information channels}

One key issue when implementing the \texttt{cfinder} compartment is that it requires an additional component to perform some “IDP proxy mapping” between node-local IDPs and remote IDPs. This issue was already identified during the development of the NetShare component (Deliverable D2.14) which is also described in [5]. It is illustrated with a simple scenario in Figure 2.

![Figure 2: IDP Proxy mapping.](image)

The scenario starts with the IPv6 FB in Node G publishing a regular expression matching IPv6 addresses in the \texttt{cfinder} compartment (reachable via IDP 'f' in Node G). This is done with the primitive call:

\begin{verbatim}
i <- publish(f, "*", "([^\x{1,4}:]{7}\x{1,4})");
\end{verbatim}
which return the IDP ’i’ via which the IPv6 FB is reachable in the cfinder compartment. We now assume that a functional block FB1 in Node A wants to communicate with the functional block FB2 in Node B. The only information it knows to setup that communication is the CONTEXT 2001::3 (i.e. the IPv6 address of Node B) and the SERVICE name FB2. We assume that FB2 has already published itself in the IPv6 compartment in Node B and is reachable via IDP ’v’. We also assume that the two cfinder FBs in nodes A and G communicate via the communication channel a→b (from Node A to Node G, the reverse channel is not shown to simplify the figure).

To initiate the communication, FB1 first needs to find a suitable compartment for the context 2001::3. It hence queries the cfinder compartment via IDP ’c’ (step 1 in the figure) with the primitive call

\[ p \leftarrow \text{resolve}(c, "\star", "2001::3"); \]

which returns the IDP ’p’, which maps an information channel to communicate with the IPv6 FB in Node G via IDP ’i’. Actually the information channel p→i already requires some IDP proxy mapping. Indeed, FB1 sends queries to IDP ’p’ while the IPv6 FB in Node G receives queries on IDP ’i’. Hence the cfinder FB in Node G must map (i.e. translate) the IDP values encoded in the queries accordingly.

To setup the information channel to FB2, FB1 can now send a resolve request to the IPv6 FB in Node G via the IDP ’p’. This is done with the primitive call (step 2 in the figure)

\[ s \leftarrow \text{resolve}(p, "2001::3", "FB2"); \]

which returns the IDP ’s’. When receiving this resolve() request, the IPv6 FB in Node G has created the information channel u→v in the IPv6 compartment. However, FB1 in Node A uses the IDP ’s’ to send to FB2 (step 3 in the figure): again, the cfinder FB in Node G must map data packets sent to the IDP ’s’ into the IDP ’u’.

In contrast to a simple forwarding operation, “IDP mapping” means that an FB must replace the IDP values encoded inside the data packets (the header of each packet is the IDP value) based on the “startpoint” IDP value (e.g. IDP ’s’ in Node A maps to IDP ’u’ in Node G). Also the IDPs encoded in the API calls need to be mapped.

It is important to note that the concept of “IDP proxy mapping” was not foreseen at the time when the modeling of ANA was developed. Hence this concept is not yet fully integrated in the current architecture, and we are currently studying how this should be properly modeled in order to nicely fit into ANA. This concept is currently used in both the cfinder and the NetShare compartments, and we envision that it could become a core and integrated feature of ANA as this seems to be a fundamental construct for advanced networking functionalities. This will be further studied in 2009.
2.3 Conclusion

As one alternative for inter-compartment routing, the \texttt{cfinder} compartment was successfully implemented in ANA. The code is available in the \texttt{C/bricks/cfinder/} directory, along with a simple template giving examples on how one should use the \texttt{cfinder} to publish regular expressions and find out which compartment handles a certain name.

In parallel to the \texttt{cfinder} compartment, a second alternative for performing inter-compartment routing was developed. A key design of ANA is indeed to offer multiple alternatives for a given task. This second design is summarized in the next section.

3 Overlay-based Inter-Compartment Communication (ANAturf)

The ANAturf compartment design is based on the TurfNet architecture described in [6] and in the ANA Deliverable D.2.3 [3], which follows the principles of overlay and underlay identifier space separation similarly to, for example, HIP. Our solution defines a global unstructured overlay identifier space which is common across network boundaries and used by all nodes that want to communicate with each other to identify source and destination. Identifiers in the underlay are defined in accordance with the underlying network protocols on top of which the ANAturf overlay is positioned. ANAturf provides a mapping between overlay and underlay identifiers. To enable communication across network boundaries, especially if different protocols are in use, address and packet translation is a necessity, which is provided by gateway nodes that are located in both networks and bridge these boundaries.

Rather than encoding location information into the address or identifier themselves, the inter-compartment routing logic provided here is based on an explicit node registration and lookup scheme in the overlay compartment through which the inter-compartment path is established. The routing at the inter-compartment level is hereby controlled primarily by the relationship among compartments (taking into account business aspects, trust, etc.), rather than by the shortest end-to-end path. This relationship can, for example, take into account whether two compartments have a customer/provider or peering relationship. As a consequence of this, the basis for inter-compartment routing is the given relationships between compartments, and the compartment hierarchy resulting from those inter-compartment relations.
3.1 Terminology

In order to simplify the description in the following section, the following terminology is introduced:

1. The term Turf or TurfNet is used to define an autonomous network (underlay compartment) which has (like any other compartment) its own identifier space for addressing and routing within the compartment. Examples of such turfs could be an IPv4 network, IPv6 network, or a switched Ethernet network.

2. A Turf Node is a network node in a specific turf. For turf-local communication, the turf node must support the local network protocols and addressing schemes. A physical node can participate as a full-edged turf node in multiple turfs at the same time, allowing multi-homing, potentially using different technologies. For communication across TurfNets turf nodes possess one or more overlay identifiers, which map into turf-local identifiers (locators) that are used for addressing and routing within the local turf.

3. For the management of the identifier space and other compartment-specific control functions (e.g. address resolution and routing), ANAturf encompass a logical entity, called Turf Control (TC). The TC provides the necessary control functions of a underlay compartment to enable inter-compartment communication based on the ANAturf overlay compartment. The TC is a compartment specific function and can depend on the actual type of underlay compartment. The TC can also be implemented in a centralized or fully distributed fashion.

4. Turf Gateways are special, multi-homed turf nodes. Besides participating in multiple turfs at the same time, they can relay traffic between these different turfs. When turfs use different addressing or protocol mechanisms, the gateways also perform the required address and protocol translations when relaying traffic. For example, a gateway between IPv4 and IPv6 turfs translates between the two network protocols and their respective address spaces. Turf gateways enable interoperation by performing the necessary translation or emulation across independent turfs, if required.

3.2 Specific design aspects

All ANAturf components (node, controller, gateway) are provided by the ANAturf brick. Whether a node is a plain node, or also provides controller and/or gateway functionality is transparent to the user. Every instance acts as a turf node, providing registration and packet forwarding functionality. The design is based on the assumption that every ANA node (i.e. every MINMEX instance) wishing to
Figure 3: Components present within the turf compartment

participate in TurfNet provides its own instance of ANAturf. Also remote utilization of the ANAturf service using other means of transport provided by the ANA framework is possible.

The turf control functionality is provided by the turf controllers, which are located on a subset of all turf nodes present within an ANA compartment. ANAturf related locator/address registration and lookup are handled transparently by the ANAturf brick on a turf node; internally these messages are forwarded to the turf controllers. Whether a node provides the controller functionality is based on the number of controllers present within the network segment and on probability. On startup (and periodically afterwards) all nodes search for neighboring controllers using broadcast messages. If no controllers are found (and also randomly based on a pre-defined probability value on startup), the node becomes a controller. Turf controllers register their presence with all neighboring controllers and transmit status update message between each other. These updates can include information about all controllers and gateways known to the controller. All ANAturf registration information is stored in a distributed hash table provided by all the connected turf controllers.

Turf nodes that are connected to more than one network segment automatically provide the gateway service. Gateway nodes take care of network address mapping and translation, registration of temporary addresses, and forwarding of packets between underlying network segments, as well as the forwarding of registrations and lookup requests to higher-level turfs (if so requested by the caller).

3.3 Communication flow example

As seen in the example depicted in Figure 4, inter-turf communication can include message flow between numerous nodes.
At startup the node A in the IP LAN registers its global overlay identifier with a local controller which forwards the registration to the higher level turf via a gateway, which allocates a mapping in the higher level turf and registers the overlay identifier in the higher level turf (see purple arrows).

Another node called B in the Ethernet turf that wants to communicate with node A sends a lookup request for the node A to a local controller. If this controller does not possess the registration information about the requested node, it forwards the request to the higher level turf, again via a gateway. The gateway node then contacts a higher level turf controller, where the registration information of node A is known. Once the gateway receives this information, it registers a mapping in the Ethernet turf and registers node A’s overlay identifier in the Ethernet turf. The original controller in the Ethernet turf then replies to node B, using the gateway’s new registration data (see yellow arrows).

Once node B receives this information, communication with node A is possible: All packets will travel through the gateways (see blue arrows), where address mapping is performed as appropriate.

### 3.4 ANA-related details and usage examples

#### 3.4.1 Overview

All client bricks wishing to use the ANAturf overlay, have to publish themselves within their node-local ANAturf brick. All communication clients connected to other ANAturf brick instances on different nodes within the overlay can then be performed by issuing a RESOLVE to ANAturf with the context field set to the overlay identifier of the node the target client’s ANAturf brick is running on.
The ANAturf bricks can connect to and utilize all ANA underlay compartments, e.g. Eth, IP, etc. and provide overlay functionality across all of these.

Overlay IDs can be of any arbitrary format as required by the user. However, to ease integration into the cfinder compartment, it is assumed that all identifiers are of the format "turf://NAME" where NAME can be any arbitrary string consisting of characters in the range [A-Za-z].

Figure 5: Communication flow and component interaction across different underlays

### 3.4.2 Startup

To enable the ANAturf functionality, it is sufficient to load the ANAturf brick into the MINMEX. The following examples are all based on the plug-in interface, other interfaces should be supported in a similar fashion. The ANAturf brick supports the following parameters:

- `n <name>`
  name: Overlay identifier of this turf node

- `l <underlay name>,<turf level>,<underlay id>[,<controller alias>]`
  underlay name: Name of this underlay compartment, e.g. eth01
  turf level: Level of this underlay in the hierarchy (only important for gateway nodes)
  underlay identifier: Underlay identifier to use for communication, e.g. IP address
  controller alias: Underlay identifier of the default controller to use

An example invocation of mxconfig to load the ANAturf brick on a gateway node:
Once loaded, the ANAturf brick tries to resolve all specified underlays and publish the service name "turf" in these. If it manages to attach to more than one underlay, the ANAturf brick will automatically enable and provide gateway functionality. Furthermore, if no controllers are found within one of the attached underlays, it will also become a turf controller.

### 3.4.3 Client-side interface

All services wishing to utilize the ANAturf functionality have to register their overlay identifiers within the ANAturf overlay compartment by sending a `PUBLISH` message to the local ANAturf brick. Registrations will automatically be forwarded to turfs on a higher level within the hierarchy. All services registered at remote ANAturf bricks within the inter-connected turfs can then be `RESOLVE`d specifying the remote overlay identifier as context. The resulting IDP can then be used to contact the remote peer’s service across network boundaries.

### 3.5 Conclusion

Just like the previously introduced `cfinder` compartment, the ANAturf compartment provides one of the inter-compartment routing schemes implemented for ANA. The code-base is functional and available in the `C/bricks/turf` directory. The provided interfaces are identical to the common interfaces of other routing compartments, enabling ANAturf to act as inter-compartment routing solution over arbitrary ANA-conforming network compartments. Thus, the examples for these other compartments can easily be adapted to support ANAturf. The implementation has been tested with the `Eth` and `IP` compartments provided as part of ANA.

### 4 Summary

This document has reported on the two implementations that have been developed in ANA to perform inter-compartment routing. These are not competing solutions: the design philosophy of ANA is to encourage variability and choice in order to offer multiple alternatives to perform a given task. Whether one solution or another should be used depends on the current networking conditions and environment.

The first solution ("Compartment Finder") presented in section 2 performs inter-compartment routing across heterogeneous namespaces without involving the use of a classical overlaying namespace where all end-systems have a unique name in the top overlay. With our solution, the top overlay only encodes namespaces
with Unix regular expressions: we assume that addresses and names are expressed in character string formats which we define as being their “public” representation. The main challenge for the implementation has been to develop a proxy service for setting information channels across multiple compartments: however thanks to the generic nature of this service, this component can be re-used by other compartments in ANA that need to offer such a proxy service.

The second solution (“ANATurf”) presented in section 3 achieves inter-compartment routing based on a classical overlaying compartment. In this solution, each entity willing to be reachable via the overlay has to publish a global unique name in the ANATurf compartment. The main advantage of this system is that it does not require any additional component such as the proxy service developed for the cfinder framework, and it offers a more classical design well in line with current practices in networking.

References


