



Human-centric algebraic machine learning

AML-IP implementation - Initial

Deliverable D6.1

WP6 - System and Tools



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952091.

Consortium



ALGEBRAIC AI



Chantalimaud
Foundation



Project

Title: ALMA: Human Centric Algebraic Machine Learning
 Acronym: ALMA
 Coordinator: Proyectos y Sistemas de Mantenimiento SL (eProsima)
 Grant agreement: 952091
 Call: H2020-EIC-FETPROACT-2019
 Program: Horizon 2020
 Start: 01 September 2020
 Duration: 48 months
 Website: <https://alma-ai.eu>
 E-mail: alma@eprosima.com
 Consortium: **Proyectos y Sistemas de Mantenimiento SL (eProsima) (EPROS)**, Spain
Champalimaud Foundation (CF), Portugal
ALGEBRAIC AI (ALG), Spain
German Research Center for Artificial Intelligence (DFKI), Germany
Inria (Inria), France
TU Kaiserslautern (TUK), Germany
FIWARE Foundation e.V. (FF), Germany
Technical Research Centre of Finland (VTT), Finland
Carlos III University (UC3M), Spain

Deliverable

Number: **D6.1**
 Title: **AML-IP implementation - Initial**
 Month: 9
 Work Package: WP6 - System and Tools
 Work Package leader: Proyectos y Sistemas de Mantenimiento SL (eProsima)
 Deliverable leader: Proyectos y Sistemas de Mantenimiento SL (eProsima)
 Deliverable type: DEM: Demonstrator, pilot, prototype
 Dissemination level: Public (PU)
 Date of submission: 2021-05-31
 Version: v2.1
 Status: Finished

Version history

Version	Date	Responsible	Author/Reviewer	Comments
v1.0	2021.05.04	EPROS	Javier París Uhryn	First Draft
v2.0	2021.05.18	EPROS	Raúl Sánchez-Mateos Lizano	Second Draft
v2.1	2021.05.28	EPROS	Raúl Sánchez-Mateos Lizano	Finished document

Executive summary

Algebraic Machine Learning (AML) is a relatively new machine learning technique based on algebraic representations of data. Unlike statistical learning, AML algorithms are robust regarding the statistical properties of the data and are parameter-free. The aim of the EU-funded ALMA project is to leverage AML properties to develop a new generation of interactive, human-centric machine learning systems. These systems are expected to reduce bias and prevent discrimination, remember what they know when they are taught something new, facilitate trust and reliability and integrate complex ethical constraints into human-artificial intelligence systems. Furthermore, they are expected to promote distributed, collaborative learning.

This report corresponds to *Deliverable D6.1 - AML-IP implementation - Initial* of the ALMA project. This deliverable describes the AML Integrating Platform (AML-IP) first design and the first prototype of the platform. The goal is developing a platform for cloud and edge computing environments. The platform will interconnect AML components (such as engines) and hook AML components with other software including data injection nodes.

Contents

Executive Summary	3
Contents	5
Acronyms	6
1 Introduction	7
1.1 Purpose of document	7
1.2 Partners involved	7
2 AML Environment Overview	7
2.1 Communication between AML nodes	8
2.2 Requirements	9
2.3 Enabling technologies	9
2.3.1 Data Distribution Service (DDS)	9
2.3.2 eProsima Fast DDS	10
2.3.3 eProsima Integration Service	11
2.3.4 miniUPnP	12
3 Design	12
3.1 AML-IP architecture description	12
3.2 Use Cases	14
3.3 AML-IP functional operation	15
3.3.1 Phases of communication	15
3.3.2 AML-IP operational area networks	15
3.3.3 AML-IP Communication in LAN	16
3.3.4 AML-IP Communication in WAN	17
3.3.5 AML-IP Communication Sequence Diagram	20
3.4 AML-IP Components	21
3.4.1 AML-IP Node	21
3.4.2 AML-IP Agent	21
3.4.3 AML-IP Discovery Server	25
3.5 Requirements targeted	25
3.5.1 Discovery through WAN	25
3.5.2 Communication through WAN	26
3.5.3 Node scalability	26
3.6 Benefits and constraints	26
4 Prototype	27
4.1 Design	27
4.1.1 Prototype Nodes	27
4.1.2 Data types	28
4.2 Experiments	30
4.2.1 LAN connectivity	30
4.2.2 WAN connectivity	31
4.3 Demonstrator	31
4.3.1 Experiment design	31
4.3.2 Download and installation	32
4.3.3 Experiment setup	33
4.3.4 Results	35

4.3.5	Experiment Conclusions	36
5	Conclusions	37
5.1	Advantages	37
5.2	Limitations	38
A	Screenshots of Demonstrator results	39
	References	42

Acronyms

AI	Artificial Intelligence.
AML	Algebraic Machine Learning.
AML-IP	AML Integrating Platform.
AML-IP-Agent	AML-IP Agent Node.
AML-IP-Alg	AML-IP Algebraic Node.
AML-IP-D	AML-IP Discovery Node.
DDS	Data Distribution Service.
DL	Description Language.
GA	Grant Agreement.
IDL	Interactive Data Language.
IGD	Internal Gateway Device.
IP	Integrating Platform.
LAN	Local Area Network.
ML	Machine Learning.
NAT	Network Address Translation.
OS	Operating System.
P2P	Peer to Peer.
PoC	Proof of Concept.
Proto-Alg-DL	AML-IP Prototype Algebraic Description Language Node.
Proto-Alg-Engine	AML-IP Prototype Algebraic Engine Node.
Proto-DiscoveryServer	AML-IP Prototype Discovery Server Node.
SHM	Shared Memory Transport.
TCP	Transmission Control Protocol.
UDP	User Datagram Protocol.
UPnP	Universal Plug and Play.
WAN	Wide Area Network.
WWW	World Wide Web.

1 Introduction

The ALMA research and innovation project aims to change the Artificial Intelligence (AI) paradigm by creating a new Machine Learning (ML) approach known as AML. AML is a new ML technique based on the algebraic representation of data. The goal of the EU-funded ALMA project is to leverage the properties of AML to develop a new generation of interactive, human-centered machine learning systems.

1.1 Purpose of document

This document describes the design patterns and specification of the AML-IP, a framework for the simple deployment of an AML architecture, i.e. one that is composed of AML computational nodes. This design has been carried out by *eProxima* based on the guidelines described in the WP6 of the ALMA Grant Agreement (GA), as well as taking into account the specifications, requirements and limitations of the AML nodes, developed by *ALGEBRAIC AI*.

To show the feasibility of the design, a prototype of the AML-IP has been implemented. This prototype, although it does not have all the intended functionality of the AML-IP, contains the key points of the communications framework so that it is possible to prove that these features of the final product are feasible prior to its implementation in a stable version. This deliverable also discusses the results obtained from the prototype implementation and the required changes and challenges to be addressed to fulfill the scope of the project.

1.2 Partners involved

Team Leader:

- Proyectos y Sistemas de Mantenimiento SL (eProxima) (EPROS), Spain

Partners:

- ALGEBRAIC AI (ALG), Spain
- FIWARE Foundation e.V. (FF)

2 AML Environment Overview

The Integrating Platform (IP) is a framework capable of stands the AML components (such as AML-IP Prototype Algebraic Engine Node (Proto-Alg-Engine) nodes, AML-IP Prototype Algebraic Description Language Node (Proto-Alg-DL) nodes or any other future component) and the intercommunication between these nodes through the Internet.

The AML environment is comprised by nodes of different kinds that communicate with each other. These nodes compute different types of data which are published and shared with other nodes in the AML ecosystem that request them.

One of AML biggest strengths is the great parallelization possibilities that its design enables, allowing different kind of nodes to share their data in order to distribute the workload. This parallelization effort works in two different ways:

- **Distributed effort:** Proto-Alg-DL nodes will create AML friendly data, and Proto-Alg-Engine nodes process this data in a completely independent way.
- **Parallel learning:** Different Proto-Alg-Engine nodes can work on the same problem independently using different training datasets, and share the knowledge learned with the other nodes working

on the same problem. This creates a more complex and accurate model. The output data that is shared is independent of the dataset being used to train the algorithm, so the training datasets can be private while the resulting knowledge could be openly published and shared with other nodes in the AML network.

2.1 Communication between AML nodes

The architecture on which this project focuses is based on the communication of different **AML nodes** through the Internet, freely distributing data and knowledge among them. The main points to be taken into account in order to properly understand AML-IP are:

- **Distributed architecture:** There is no main access point to the network, nor is there a centralised server to provide communication. This means Peer to Peer (P2P) communication is required between different AML nodes.
- **Data Distribution Service (DDS):** DDS will be the main communication protocol between nodes of the network.
- **World Wide Web (WWW):** This architecture is deployed over the Internet, allowing anyone with internet access to be part of the network.

Figure 1 presents a trivial example of an AML execution. There are several nodes along the network that works under a DDS topic. Each node can publish under these DDS topics and/or subscribe to them, communicating with all other nodes along the AML network working on these same topics. Thus a P2P communication is achieved in which communication is not centralised.

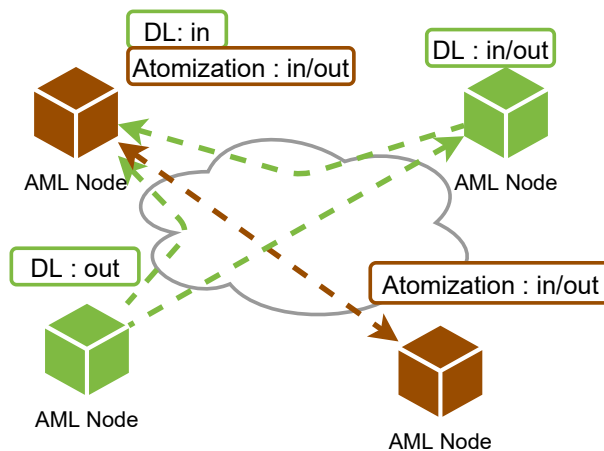


Figure 1: AML Nodes communicating with each other.

In the scenario shown in Figure 1, two types of AML nodes are presented. The green nodes communicate through the DL topic. This topic stands for the Proto-Alg-DL, the language used by all AML nodes to communicate with each other. Moreover, the brown nodes represent AML computational engines. They produce a different type of data known as atomization. This atomization data can be either the input or the output of the node. Therefore these nodes can both publish this data type under the Atomization DDS topic or/and subscribe to it.

2.2 Requirements

This section describes the high-level requirements that the AML-IP must meet in order to achieve the functionality expected from such a communications framework.

- **Decentralized Communication** Nodes must be able to communicate to each other without an intermediate communication server.
- **Publish/Subscribe Communication** Nodes must be able to communicate their data using DDS, a data-centric communication protocol used for distributed software application communications.
- **Logical Partition** Nodes must be able to communicate only with the nodes that required their information, avoiding to send data to nodes that do not require it.
- **Connection through Local Area Network (LAN)** Nodes must be able to communicate with different nodes running in the same LAN.
- **Connection through Wide Area Network (WAN)** Nodes must be able to communicate with different nodes running in various LAN, i.e. deployed in a WAN.
- **Dynamic discovery** The network must allow the creation or destruction of nodes along the execution.
- **Data Injection Nodes** AML is a growing project and hence there the platform must facilitate the addition of new types of nodes with new data formats along the project.

2.3 Enabling technologies

This section describes the different technologies that support the development of the AML-IP. All of them focus on the communication between nodes, the protocols used to support such communication and the tools used to handle the different types of data to be transmitted.

2.3.1 Data Distribution Service (DDS)

Given the distributed nature of the AML ecosystem, the use of DDS is proposed to enable real-time communication between AML nodes. As explained in the eProxima Fast DDS documentation [1]:

The Data Distribution Service (DDS) is a data-centric communication protocol used for distributed software application communications. It describes the communications Application Programming Interfaces (APIs) and Communication Semantics that enable communication between data providers and data consumers.

Since it is a Data-Centric Publish Subscribe (DCPS) model, three key application entities are defined in its implementation: publication entities, which define the information-generating objects and their properties; subscription entities, which define the information-consuming objects and their properties; and configuration entities that define the types of information that are transmitted as topics, and create the publisher and subscriber with its Quality of Service (QoS) properties, ensuring the correct performance of the above entities.

DDS uses QoS to define the behavioral characteristics of DDS Entities. QoS are comprised of individual QoS policies (objects of type deriving from QoSPolicy).

The different entities involved in the DDS protocol are also defined in [1].

In the DCPS model, four basic elements are defined for the development of a system of communicating applications.

- *Publisher.* It is the DCPS entity in charge of the creation and configuration of the DataWriters it implements. The DataWriter is the entity in charge of the actual publication of the messages. Each one will have an assigned Topic under which the messages are published.
- *Subscriber.* It is the DCPS Entity in charge of receiving the data published under the topics to which it subscribes. It serves one or more DataReader objects, which are responsible for communicating the availability of new data to the application.
- *Topic.* It is the entity that binds publications and subscriptions. It is unique within a DDS domain. Through the TopicDescription, it allows the uniformity of data types of publications and subscriptions. See Topic for further details.
- *Domain.* This is the concept used to link all publishers and subscribers, belonging to one or more applications, which exchange data under different topics. These individual applications that participate in a domain are called DomainParticipant. The DDS Domain is identified by a domain ID. The DomainParticipant defines the domain ID to specify the DDS domain to which it belongs. Two DomainParticipants with different IDs are not aware of each other's presence in the network. Hence, several communication channels can be created. This is applied in scenarios where several DDS applications are involved, with their respective DomainParticipants communicating with each other, but these applications must not interfere. The DomainParticipant acts as a container for other DCPS Entities, acts as a factory for Publisher, Subscriber and Topic Entities, and provides administrative services in the domain. See Domain for further details.

The dynamic addition of new entities to the network is also covered by the the DDS protocol, allowing the dynamic discovery of new instantiated DDS entities. The Discovery Protocol is the mechanism that DDS provides for the discovery of new entities.

Thus, DDS fulfilled various communication requirements of the AML outlined in Section 2.2. However, the WAN communication, and specially the WAN dynamic discovery of newly instantiated DDS entities is a great challenge to DDS, as this protocol relies on multicast communication. The growing type of nodes and data formats makes a challenge in order to achieve fast DDS deployments over WAN.

2.3.2 eProxima Fast DDS

As described in the *Fast DDS* documentation [2]:

eProxima Fast DDS is a C++ implementation of the DDS (Data Distribution Service) Specification, a protocol defined by the Object Management Group (OMG). The eProxima Fast DDS library provides both an Application Programming Interface (API) and a communication protocol that deploy a Data-Centric Publisher-Subscriber (DCPS) model, with the purpose of establishing efficient and reliable information distribution among Real-Time Systems. eProxima Fast DDS is predictable, scalable, flexible, and efficient in resource handling. For meeting these requirements, it makes use of typed interfaces and hinges on a many-to-many distributed network paradigm that neatly allows separation of the publisher and subscriber sides of the communication.

Among the many features of *Fast DDS*, there are two key functionalities for the development of the AML-IP: a) the TCP transport, and b) the Discovery Server discovery mechanism.

Among the multiple transport protocols and communication mechanisms available in *Fast DDS*, the TCP transport protocol is implemented for data transmission between DDS entities in WAN networks. However, the configuration of DDS entities to use this protocol is not straightforward at this point. The purpose of the AML-IP is to implement a communications framework that uses the Transmission Control

Protocol (TCP) communications protocol implemented in *Fast DDS* in a transparent manner for the user.

Moreover, according to [3]:

Fast DDS, as a Data Distribution Service (DDS) implementation, provides discovery mechanisms that allow for automatically finding and matching DataWriters and DataReaders across DomainParticipants so they can start sharing data.

The AML-IP framework will exploit the Discovery Server discovery mechanism described in [3] as follows:

This mechanism [Discovery Server] is based on a client-server discovery paradigm, i.e. the metatraffic (message exchange among DomainParticipants to identify each other) is managed by one or several server DomainParticipants [...], as opposed to simple discovery [...], where metatraffic is exchanged using a message broadcast mechanism like an IP multicast protocol.

The *Fast DDS* Discovery Server is thought to reduce the traffic network and to assist in DDS discovery in cases where multicast is not an option in a dynamically scalable DDS networks. Moreover, it centralizes the discovery information for the *Fast DDS* DomainParticipants connected to it.

There are some key aspects that need to be covered in order to understand the Discovery Server discovery mechanism. The following can be gleaned from the *Fast DDS* online documentation [3]:

- *Discovery Server DomainParticipants may be clients or servers. The only difference between them is on how they handle discovery traffic. The user traffic [...] is role-independent.*
[...]
- *All server and client discovery information will be shared with linked clients. [...] a server may act as a client for other servers.*
- *A **SERVER** is a participant to which the clients (and maybe other servers) send their discovery information. The role of the server is to re-distribute the clients (and servers) discovery information to their known clients and servers. A server may connect to other servers to receive information about their clients. Known servers will receive all the information known by the server. Known clients will only receive the information they need to establish communication, i.e. the information about the DomainParticipants, DataWriters, and DataReaders to which they match. This means that the server runs a “matching” algorithm to sort out which information is required by which client.*
[...]
- *A **CLIENT** is a participant that connects to one or more servers from which it receives only the discovery information they require to establish communication with matching endpoints.*

2.3.3 eProsima Integration Service

eProsima Integration Service is a tool for the interconnection of nodes that communicate using different protocols. This is done by translating the protocol used at one endpoint to a common protocol internal to the Integration Service tool. Currently, five protocols are supported allowing communication between all of them.

The development path to follow is based on the enhancement of this tool so that, in addition to being used as a protocol translator, it supports the easy deployment of applications over WAN networks.

2.3.4 miniUPnP

Due to the distributed nature of the AML environment, nodes require a mechanism to be accessed from outside their LAN. This is made possible by port forwarding of the user's private Internal Gateway Device (IGD) using the Universal Plug and Play (UPnP) protocol.

UPnP is a set of protocols that allow network devices to perform actions for automatic discovery of services on the network. In the specific case of ALMA, this protocol is used for the user's IGD to establish a communication channel between AML-IP elements that are deployed in other networks external to the user's private network (LAN). This is done by port forwarding on the router, i.e., pairing the public address of the IGD and a public port with the private IP-port pair of the machine running the AML-IP node.

The implementation of the UPnP protocol in the AML-IP elements is done using the miniUPnP library. miniUPnP is a C library that supports the UPnP IGD specifications. The purpose is to provide AML-IP nodes with the capabilities to perform this action automatically and transparently to the user.

3 Design

In this section the AML-IP design is presented and discussed. This does not intend to be a final product design, but a dynamic and living design that fulfills the actual project requirements and handles future scalability and new requirements.

3.1 AML-IP architecture description

The approach this design has chosen is focused on the division of the different AML-IP components that constitute the network. In order to divide the computational effort, to distribute the communication process and to create an scalable and widely usable architecture, the components of the AML-IP are divided in three categories, i.e. the AML-IP architecture consists of three types of nodes:

- **AML-IP Algebraic Node (AML-IP-Alg):** These nodes represent each of the instances of the AML.
- **AML-IP Agent Node (AML-IP-Agent):** This is the main transport component that manages the dynamic discovery of AML-IP Algebraic Nodes (AML-IP-Algs) deployed within a LAN and the discovery of other AML clusters over WAN.
- **AML-IP Discovery Node (AML-IP-D):** This component will be disseminated along the network and accessible through the Internet. They manage the WAN discovery process along the whole AML environment, handling the discovery of independent AML clusters and efficiently connecting the AML-IP-Alg that must connect to each other.

The layer model of the AML-IP is shown in Figure 2. This shows how the AML-IP would integrate all transport and network elements, as well as all types of AML-IP nodes.

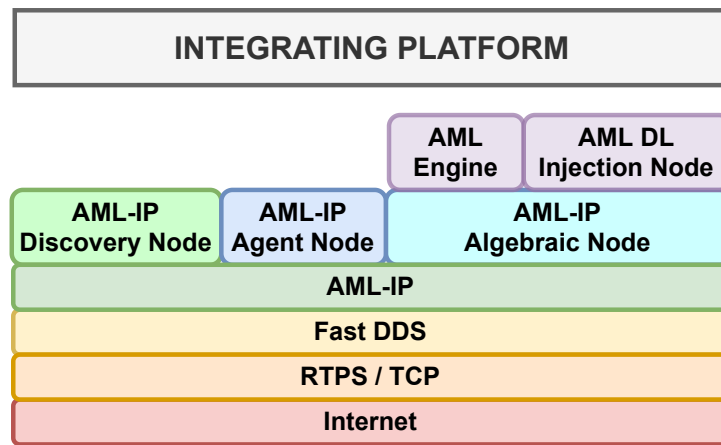


Figure 2: AML-IP layer model.

Figure 3 shows a general example of an AML network and how the AML-IP will establish a communication method for the distributed nodes. The AML-IP Discovery Node (AML-IP-D) are network nodes that are distributed along the Internet and work as a *well known address* in order to manage the WAN discovery. Other kind of nodes are the AML-IP Agent Node (AML-IP-Agent), network nodes deployed by each user. They manage the connection between a LAN cluster and the rest of the network. The AML cluster is just a group of nodes inside the same LAN that can communicate with each other within the LAN to which they belong.

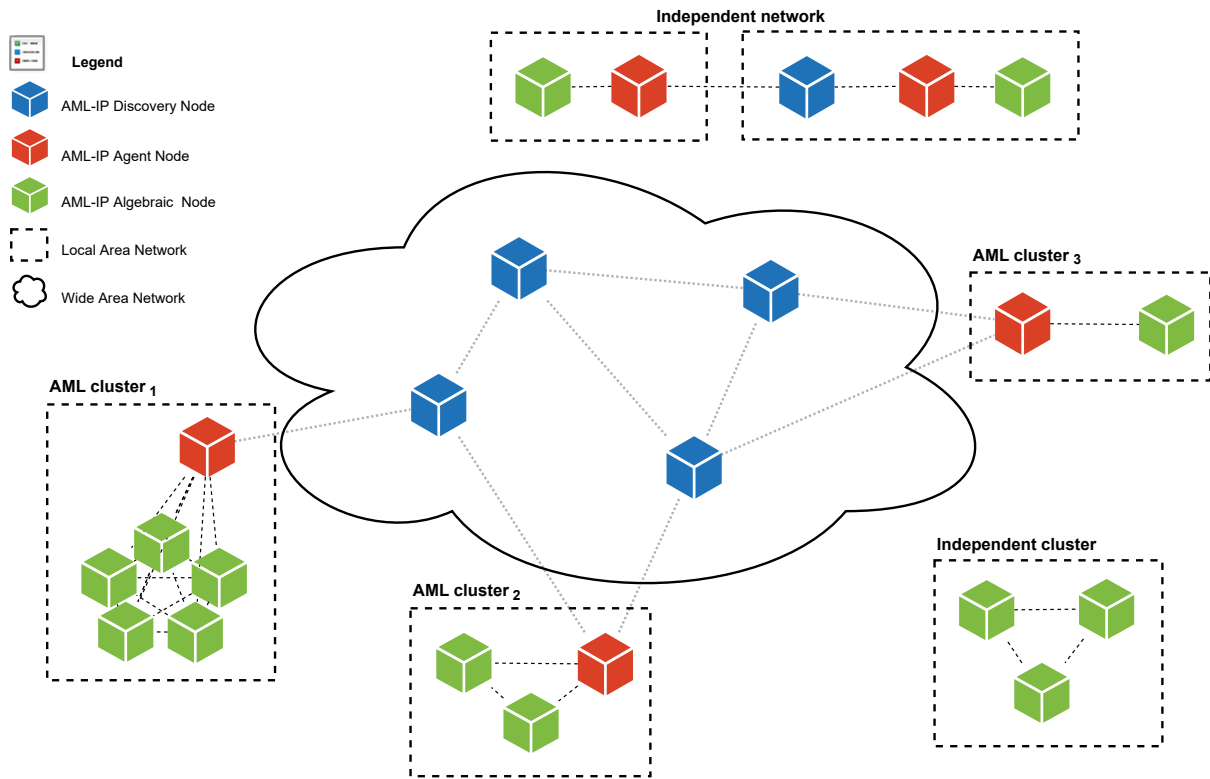


Figure 3: AML-IP example in LAN and WAN networks.

3.2 Use Cases

The AML-IP is a communication framework focused on the easily deployment of AML nodes and the communication between them. Thus, AML-IP is not just an user oriented application, but an intermediate library that makes the transport protocols transparent to the user. Therefore, it is thought to create a platform that communicates each node with those that are in the same AML network without requiring the user to know or understand the transport protocols behind each communication.

The AML-IP users are divided in two categories represented in the figure 4. This diagram represents only the actions of each user with the AML-IP, i.e. it does not represent the actions that a user could do with each of the AML-IP-Algs.

The first actor shown, referred to as the AML User, represents the standard AML user. This user in charge of the LAN where it deploys its AML nodes. This user would be able to create any kind of AML-IP-Alg within its own LAN, and to start a new AML-IP-Agent in order to connect with an external AML network deployed in another LAN, creating a wider AML environment over WAN. Starting an AML-IP-Agent may require to set one or more *well known addresses* in order to connect with a bigger network.

The second actor is the AML-IP Administrator. This kind of actor represents a platform administrator with every level of permissions over the networks. In addition to the actions that a *User* can do, the *Administrator* is able to deploy AML-IP-D in order to increase the range and redundancy of a network.

All types of nodes as well as the different architectures that may exist in an AML environment are

described in Section 3.1.

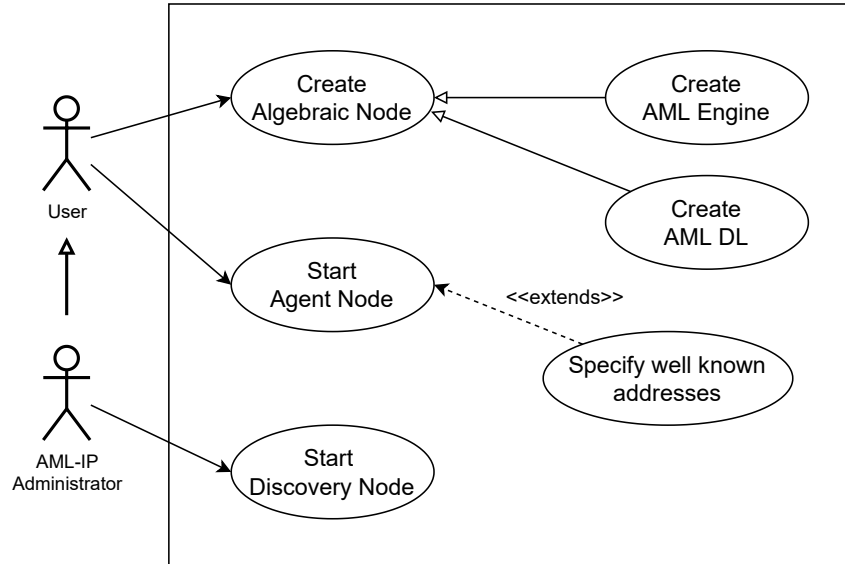


Figure 4: Use case diagram.

3.3 AML-IP functional operation

This section explains and exemplify the communication process of all nodes within the the AML-IP. The differences between the various stages of inter-node communication are also explained. Finally, the type of communication is distinguished according to the location of the nodes in the network, i.e. whether the nodes belong to the same local network (LAN) or whether they are located in different networks (WAN).

3.3.1 Phases of communication

The communication process in DDS is divided in two phases: the discovery phase and the data transfer phase:

- **Discovery phase:** The discovery phase establishes the mechanism by which new DDS entities are detected by other entities already operating in the network. New entities communicate their arrival in the network via messages to a multicast group, and continue sending these messages in order to discover new entities or inform known entities of their current status. These messages are required to maintain the network state in a dynamic and distributed way.
- **AML data transfer phase:** This phase corresponds to the transfer of data required by the AML-IP-Alg to correctly execute their function.

3.3.2 AML-IP operational area networks

The networks where the AML-IP could be operate are the Local Area Network (LAN), whether all the hosts are under the same switch or router and share a private network; or the Wide Area Network (WAN), whether nodes are distributed along a wide network and usually behind different Network Address Translations (NATs).

In the figure 5 it is represented an example of communication between different kinds of nodes deployed in LAN and WAN. The discovery and the data transfer phases depends on the network type. The node behaviour in each network kind are explained in the following sections.

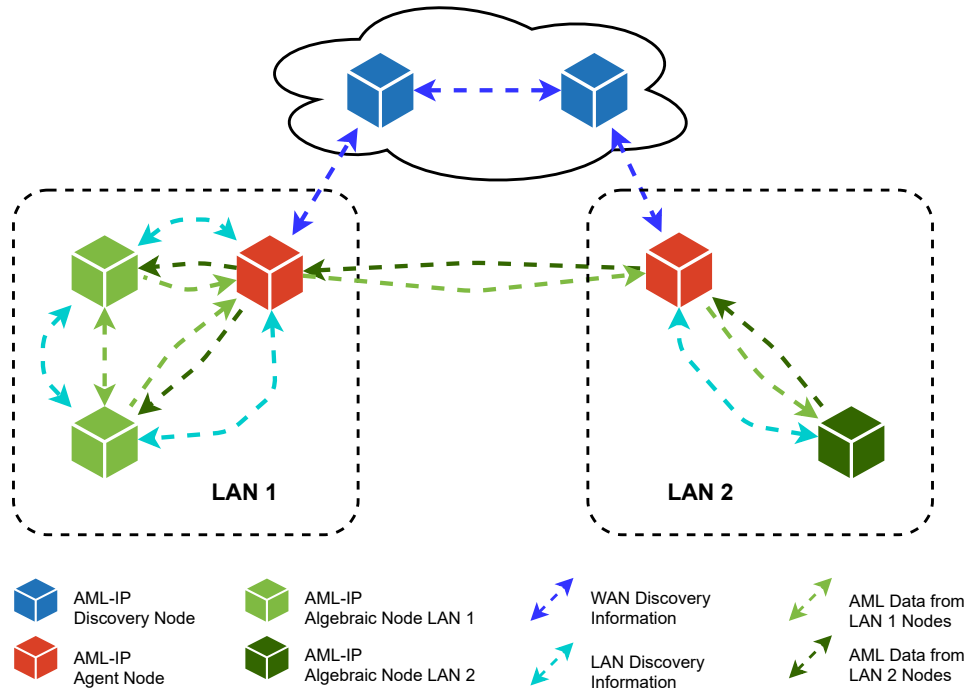


Figure 5: AML-IP minimum example to describe LAN and WAN communication.

3.3.3 AML-IP Communication in LAN

In LAN communication each node will discover each other by publishing and listening in a multicast group inside the network. In computer networking, multicast is a group communication that enables the data transmission to multiple targets [4].

The discovery mechanism used in LAN communication is called *Simple Discovery Protocol* [2], the standard protocol for dynamic discovery in DDS. This way, every AML-IP-Alg in the network will know each other, and they will discover an instance of AML-IP-Agent in case it exists. Once the discovery has been established, each node will communicate with each other directly, as long as one node is publishing and the other subscribed to the same shared topic. The AML-IP agent will subscribe to each topic existing in the network, and it will collect the data published in them and resend it through WAN.

Figure 6 represented an easy example on how the LAN communication will operate. The top and middle network architectures (steps 1 and 2) of Figure 6 show the discovery phase between all the components of the AML network, which are the AML-IP-Alg in this case. Assuming that they all belong to the same multicast group, they will send their data to the other network nodes in that multicast group, and in turn they will all receive the information from the other nodes in that group. The network architecture at the bottom of Figure 6 (step 3) shows the data transfer phase, which is performed directly between the AML-IP-Alg that have been matched in the previous discovery phase.

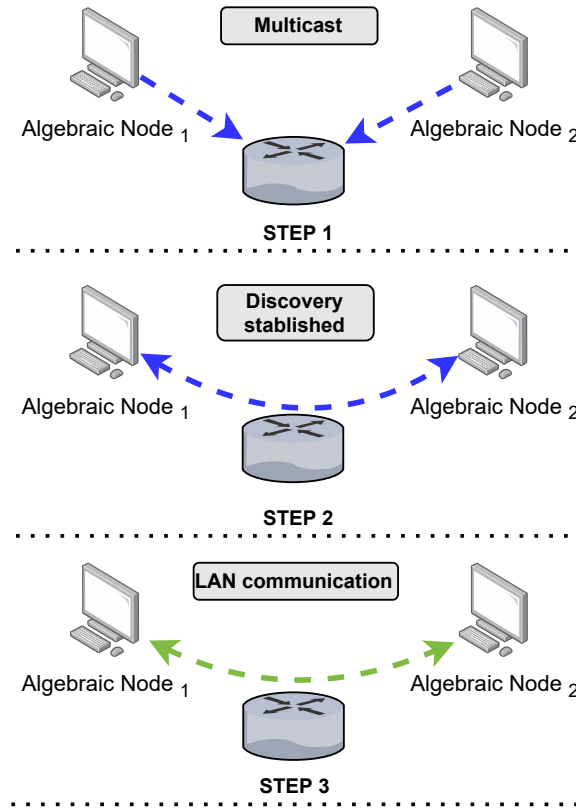


Figure 6: LAN communication schema.

3.3.4 AML-IP Communication in WAN

In WAN communication, the discovery protocol used is different than in LAN. Due to the impossibility of using multicast in WAN, the discovery mechanism must be centralized in an AML-IP node referred to as AML-IP-D. Connecting AML-IP-Agent to an AML-IP-D will make this node to know every other node in the network operating under the same topics, as well as every other AML-IP-Agents already deployed will know the new one.

A simple example of WAN communication is shown in figure 7. Once an AML-IP-Agent connects to the AML-IP-D it receives all the necessary information to communicate P2P with the rest of the nodes. The first step shows a stable network architecture in which there are two agent nodes, referred to as *AgentNode₁* and *AgentNode₂*, that have already been matched and know each other, and the appearance in the network of a new *AgentNode₃*. Moreover, during this first stage, *AgentNode₃* sends its discovery information to *DiscoveryNode₁*. In the second stage, *DiscoveryNode₁* sends to *AgentNode₁* and *AgentNode₂* the discovery information of *AgentNode₃*, and to *AgentNode₃* the discovery information of the first two agent nodes. Finally, in stage 3, all nodes have been matched and are ready to exchange AML data with each other.

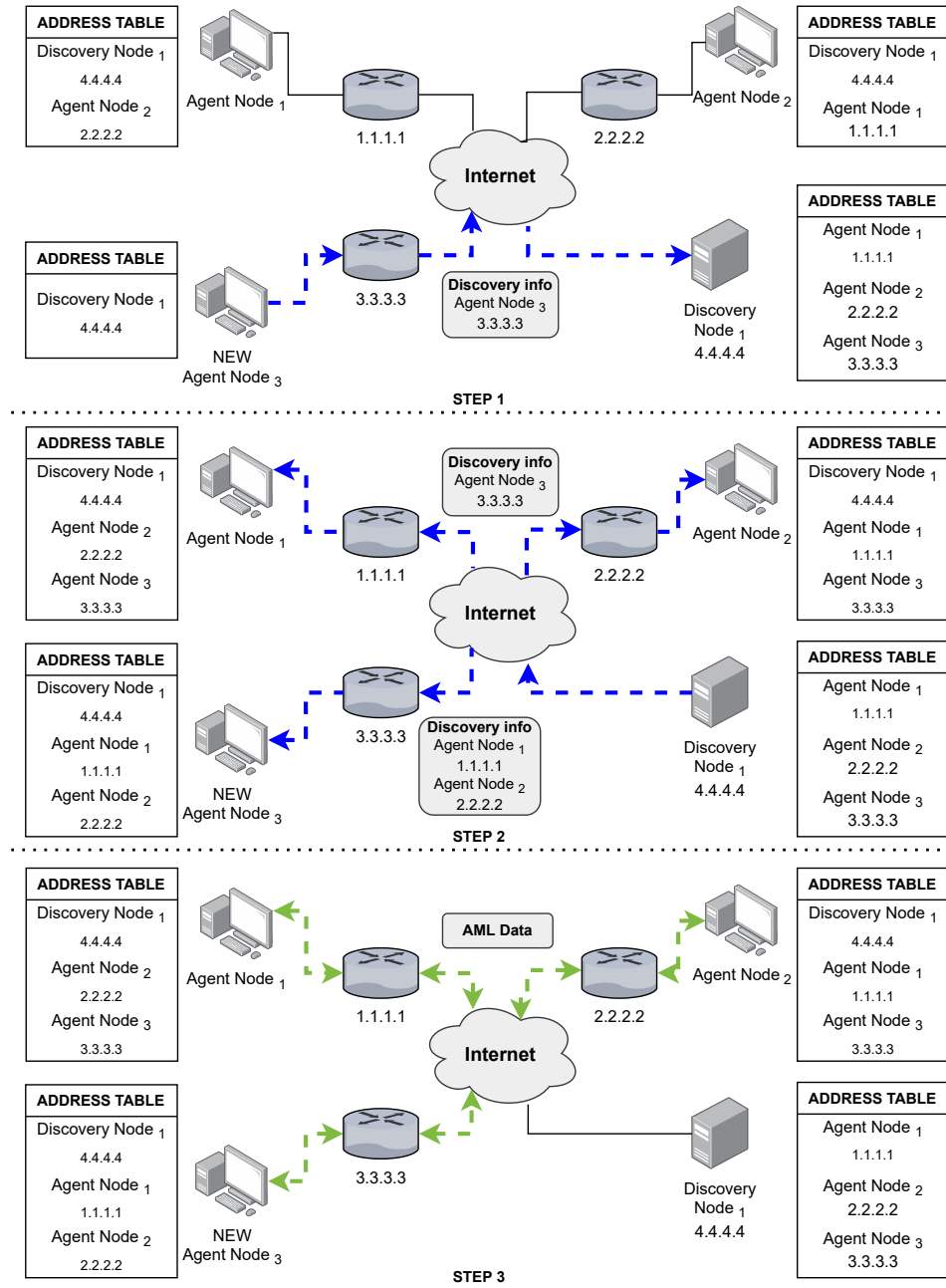


Figure 7: WAN communication schema.

These AML-IP-Ds allow to create an interconnected network that distributes the discovery information from one AML-IP-Ds to another. There is an example on how a network with two AML-IP-Ds works in Figure 8. In stage 1 shown in Figure 8, a WAN network architecture is shown in which there is one agent node ($AgentNode_1$) and two discovery nodes deployed, $DiscoveryNode_1$ and $DiscoveryNode_2$. In this same stage a new agent node ($AgentNode_2$) is added to the network which sends its discovery information to a single discovery node ($DiscoveryNode_1$) previously configured. In stage 2, $DiscoveryNode_1$

forwards the discovery information of *AgentNode₂* to *DiscoveryNode₁*, and to *AgentNode₂* the discovery information of the rest of the agent nodes it knows, which in this case is just the discovery information of *AgentNode₁*. In phase 3, which can occur in parallel in time to phase 2, *DiscoveryNode₂* sends the discovery information of *AgentNode₂* that it has just received to *AgentNode₁*. This phase ends with all agent nodes matched. Finally, since all agent nodes know each other, the P2P data transfer phase begins.

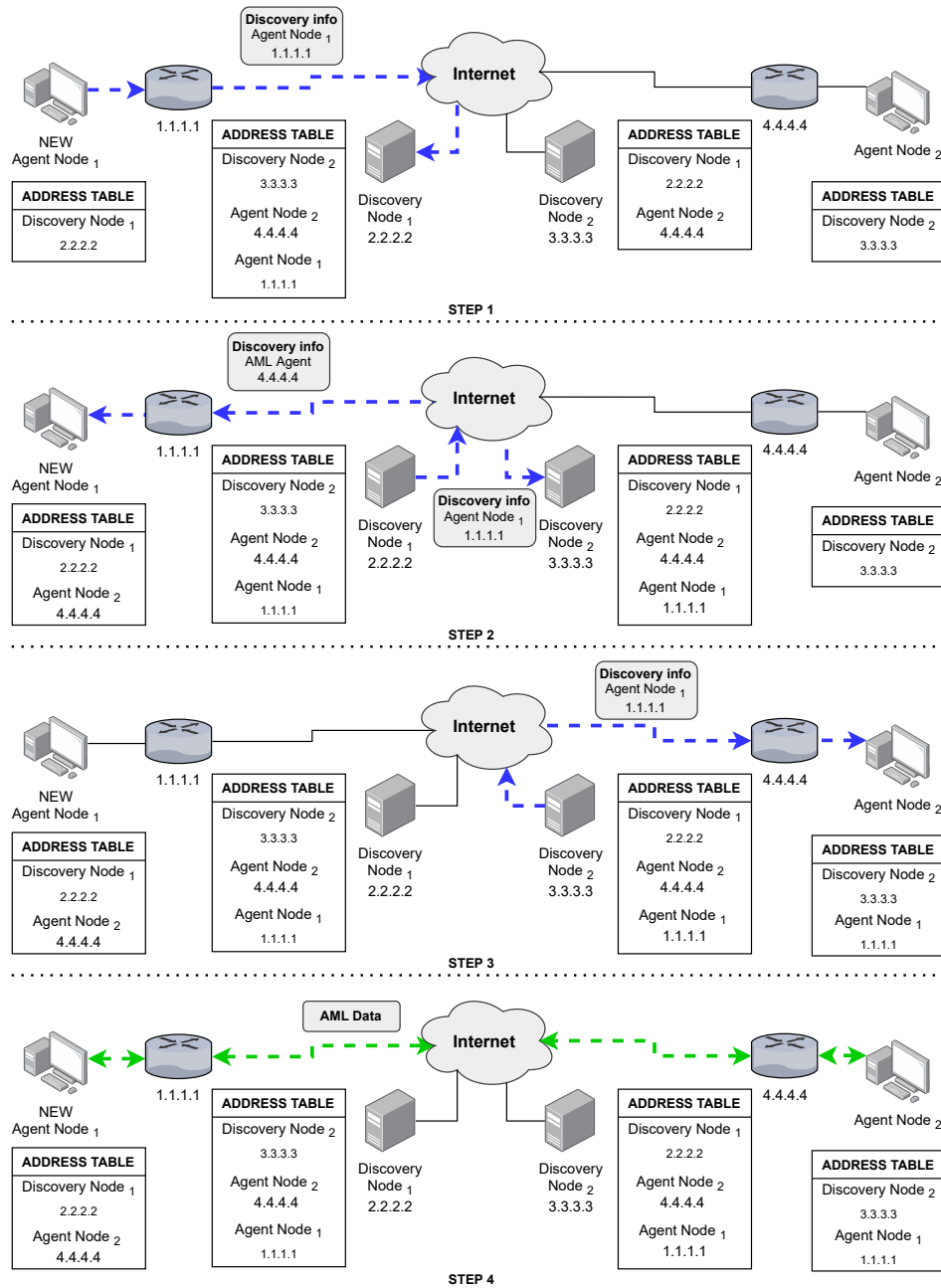


Figure 8: WAN communication schema with a network of AML-IP-D.

The use of multiple AML-IP-Ds introduces the following enhancements to the AML-IP architecture:

- Increase the number of *well known addresses* that an AML network could have.
- Enhance the robust of the network since when one AML-IP-D has dropped, the others will still handle the discovery phase.
- Allow the redundancy of discovery information to avoid possible network issues.
- Distribute computational effort in large networks.

3.3.5 AML-IP Communication Sequence Diagram

A sequence diagram of how the communication process is managed by the AML-IP is shown in Figure 9. First, an AML-IP-Agent should be started in the network in order to allow WAN communication. This AML-IP-Agent will send its own discovery information to its *well known addresses* in order to stablish the communication with the rest of the network. Besides that, and not necessarily after it, the AML-IP-Agent will know every AML-IP-Alg in its own LAN by using multicast communication. Once all the discovery has been settled, the communication from one AML-IP-Alg to the other will be performed by communicating to their respective AML-IP-Agent, and those communicating to each other.

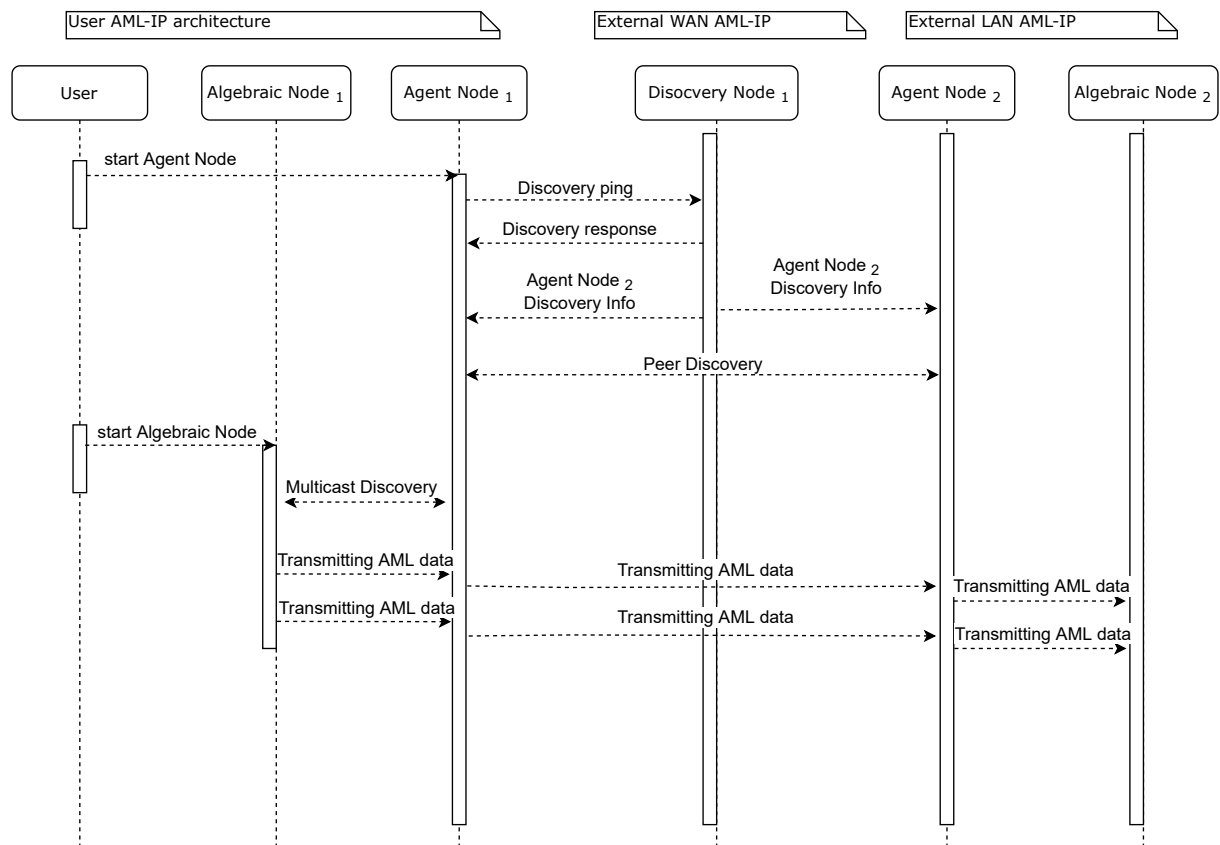


Figure 9: AML-IP sequence diagram in LAN and WAN networks.

3.4 AML-IP Components

As stated in 3.1, there are three types of AML-IP nodes: *a)* AML-IP Algebraic Node (AML-IP-Alg), *b)* AML-IP Agent Node (AML-IP-Agent), and *c)* AML-IP Discovery Node (AML-IP-D). This section describes in detail each of them as well as the benefits and solutions offered by each of them to the challenges listed in Section 2.

3.4.1 AML-IP Node

The AML-IP-Alg is the main node of an AML execution. It is where the AML algorithm and all the data it requires will be executed. An example on how these nodes will be implemented could be seen in Figure 10.

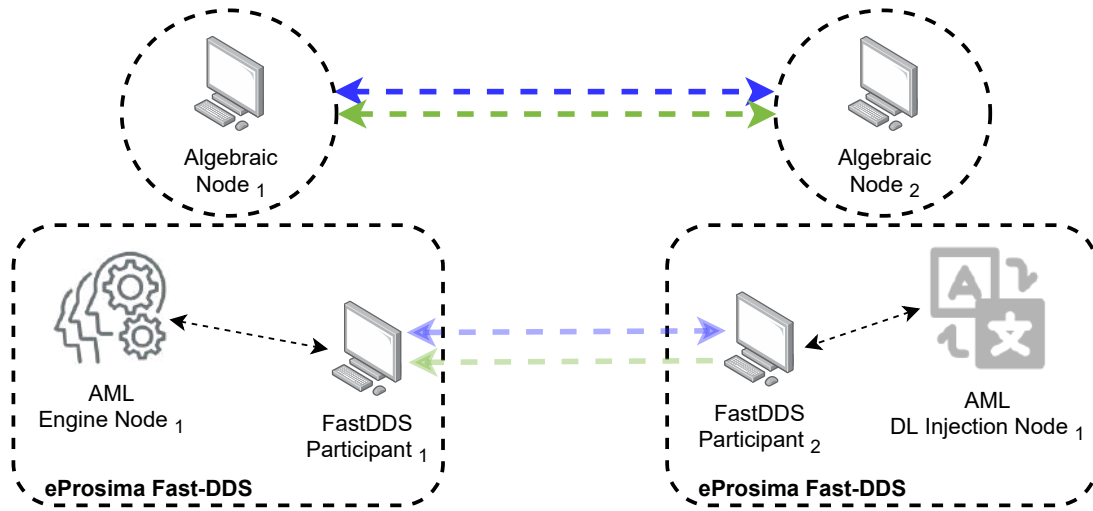


Figure 10: AML-IP Algebraic Node (AML-IP-Alg) design.

The AML-IP-Alg is an abstraction of every node required by the AML algorithm to be executed. Each kind of these nodes will send or receive under a topic that could affect to none, one or multiple remote AML-IP-Alg. This information will be distributed relying on the DDS standard to every node that requires it, using specific topics for each kind of data and each problem on which the AML node is working. *Fast DDS* will be used as the C++ library that implements the DDS communications standard. *Fast DDS* is easy to integrate in every application, and it is an Open Source project, what makes it the best candidate to be part of every AML node. Section 2.3.2 describes *Fast DDS* in detail.

The design, implementation and operation of the AML nodes is not part of the *Deliverable D6.1 - AML-IP implementation - Initial*, and it is led by *ALG*. The integration and deployment of these nodes with the AML-IP will be made by *EPROS* and *ALG* collaboratively.

3.4.2 AML-IP Agent

The AML-IP-Agent is the node in charge of communicating a local node or AML-IP cluster with the rest of the network in WANs. It centralizes the WAN discovery and communication, i.e. it is the bridge for all the nodes in their LANs with the rest of the AML-IP components. The key functionalities of the AML-IP-Agent are *a)* the Data Broker, *b)* the discovery of AML-IP components in WAN, and *c)* the mechanisms to make the AML-IP components accessible from outside the LAN using UPnP.

Internally, the AML-IP-Agent will use an *eProxima* product called **Integration Service**. This tool is developed and maintained by *eProxima* which facilitates certain DDS behaviours, as the WAN or the UPnP administration. This tool allows to create a bridge between two *DDS DomainParticipants*, which fill the gap between the LAN and WAN. The main benefits of using the Integration Service tool are the following:

- It will facilitate the configuration of WAN communication, making it unnoticed for an AML user.
- It will be improved with UPnP functionalities in order to automatically administrate the UPnP connections and the upload of its discovery information.
- It will be improved with the **eProxima Fast DDS Discovery Server** which will automatically create a *Discovery Server* in client mode to share its discovery information and to get other nodes information from a AML-IP-D.

Figure 11 shows these two *DDS DomainParticipants* that comprise the AML-IP-Agent and the communication process in a WAN scenario. This figure also shows how the AML-IP-Agent acts as a central connection node in a LAN to connect AML-IP-Alg nodes in WAN. On the left side of Figure 11, it can be seen that AML-IP-Alg nodes internal to a LAN (*AlgebraicNode₁* and *AlgebraicNode₂*) discover each other and in turn automatically discover the *AgentNode₁* present in their LAN (blue dots line). Once these nodes have been discovered, the *AlgebraicNode₁* and *AlgebraicNode₂* start the AML data transfer phase within the LAN in case they are both matched, i.e. both are working under the same topic. Furthermore, all data to be published is also transmitted to *AgentNode₁*, which acts as a data broker. At this point in the communication, the *AgentNode₁* node knows the AML-IP-Alg nodes internal to its LAN, and all the data they publish outside the LAN are transmitted to it. Then, to find out if there are more AML-IP-Alg in the WAN, the *AgentNode₁* node uses its *Discovery Server client*, referred to as *FastDDSParticipant₂* in this case, to initiate the discovery process of other AML-IP-Agent and AML-IP-Alg nodes in the WAN. This process is shown in Figure 7.

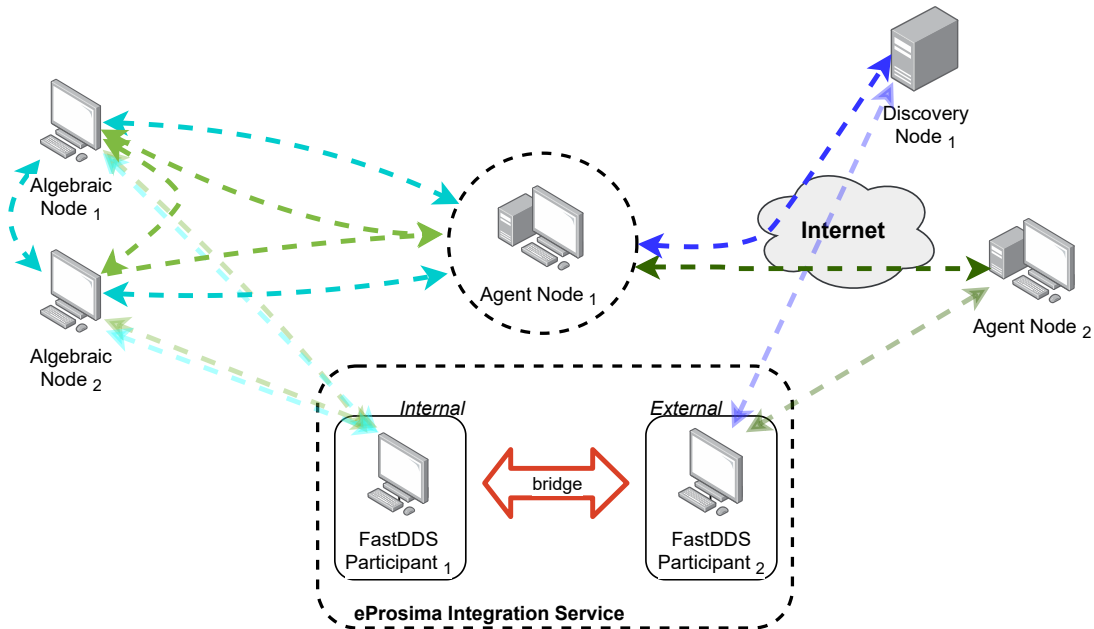


Figure 11: AML-IP Agent Node design.

a) Data Broker

The AML-IP-Agent centralizes the communication in the LAN to be sent and received in WAN. That is, every message that needs to be sent outside the LAN or to be received from outside must pass through it.

In order to send data in WAN, the *Internal Participant* subscribes to every topic in its LAN and receives every message to pass it forward to the *External Participant*. The *External Participant* publishes this message to every other AML-IP-Agent subscribed in that topic in WAN.

The method to receive messages from outside the WAN is similar. The *External Participant* subscribes to every topic in WAN that is relevant to its LAN. That is, it subscribes to every topic that has a subscriber in any internal AML node. Therefore, it will receive from other AML-IP-Agents the messages regarding this topic, and it will send it backward to the *Internal Participant*. Finally, this *Internal Participant* will publish these messages received in WAN under the same topic in LAN, making it possible for all nodes subscribed to this topic to receive the information.

b) **WAN discovery**

The AML-IP-Agent has an internal AML-IP-D that connects as a *Client* with one or multiple AML-IP-Ds, setting their addresses beforehand. As it is shown in Figure 11, the Integration Service will connect with an AML-IP-D, uploading the required internal discovery information to that AML-IP-D and downloading the whole discovery information from the same AML-IP-D.

c) **NAT transversal**

One of the main challenges that a P2P connection in WAN poses is how to communicate two nodes that are behind NATs. NAT is an IP address translation mechanism whereby one address space is converted to another address space different from the first. The vast majority of NATs convert a single public IP address into a private IP address space. This implies a difficulty, or even an impossibility in certain types of NAT (symmetrical NAT [5]), to communicate two nodes behind different NATs without specific router configurations.

In order to overcome this problem, the way chosen is to use UPnP, a well known set of networking protocols that allows to automatically apply a configuration on the router in order to allow external connections to reach a specific internal node. UPnP will be used by the AML-IP-Agent to "open" or forward a public address to a specific private address behind the NAT. This allows to another AML-IP-Agent in the Internet to communicate directly with an AML-IP-Agent deployed behind a NAT knowing its public address and the port forwarded.

Figure 12 shows an example of how an AML-IP-Agent uses UPnP to be accessible to other AML-IP-Agents deployed in other LANs. Moreover, the UPnP has a mechanism to eliminate the forwarded ports when a period of time has expired. This mechanism will be exploited by the AML-IP-Agents as there is no need to keep the port forwarded once the connection has been established.

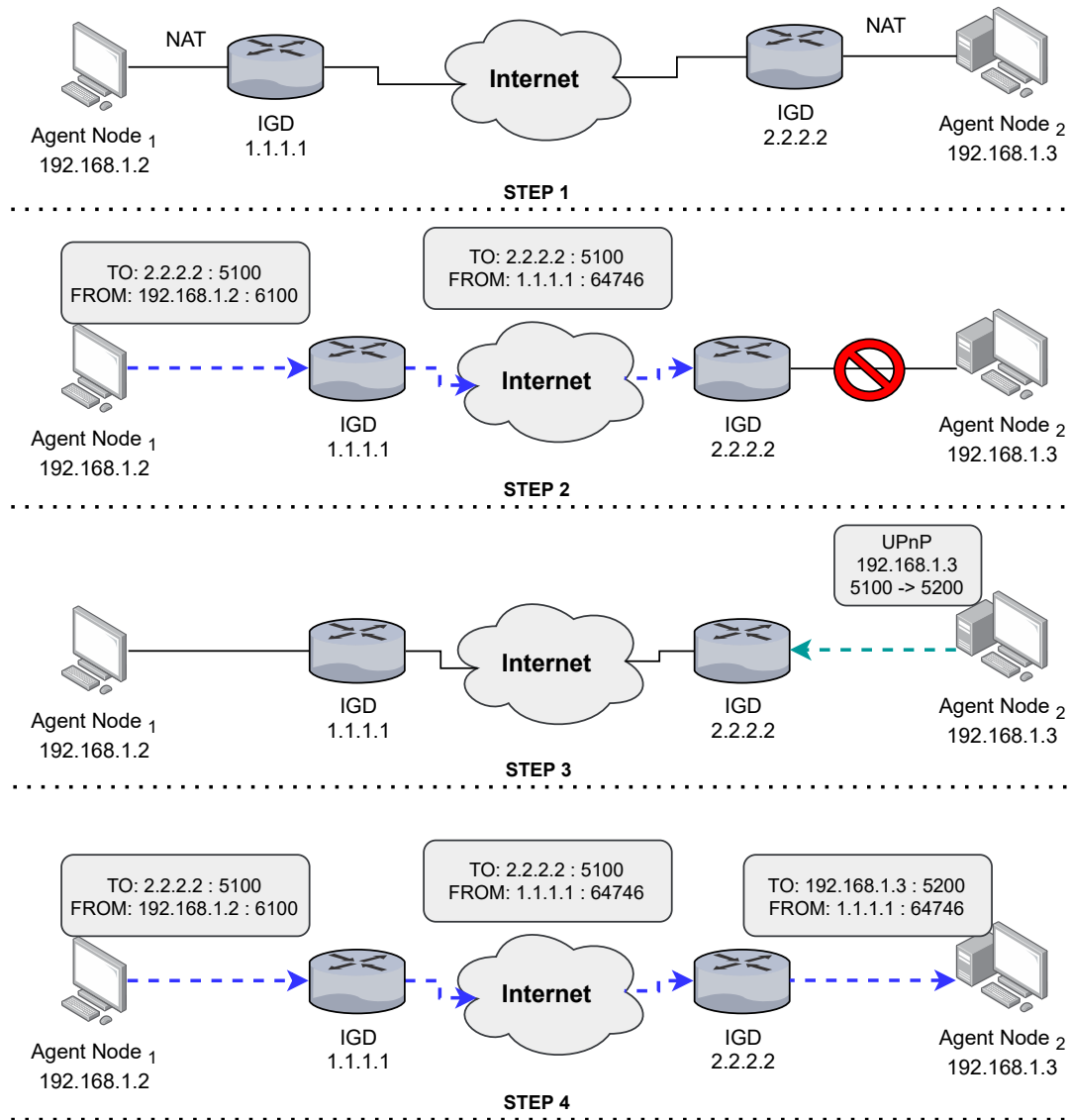


Figure 12: UPnP execution example.

One difficulty of the UPnP is to manage which ports are opened by which node, and to maintain a coherence and the security of a network with open ports. For this reason, the UPnP administration will be done by the AML-IP-Agent, and it will be the only one with open access from the internet. Each AML-IP-Agent will open its own port in the router and the rest of the AML-IP-Agent will communicate to this agent by its public address and the port forwarded, and this data is known by all the agents because of the discovery information distributed by the AML-IP-D. This way, an attack would not affect an AML node but just a communication node, and the UPnP administration would be centralized in a single entity.

3.4.3 AML-IP Discovery Server

The AML-IP-D is the AML-IP node in charge of receive, redistribute and maintain the discovery information of the whole network. The AML-IP-D could create networks by connecting AML-IP clusters and redistributing all the discovery information between them. Thus, it could be used to build a large and redundant network, in which a single AML-IP component gives access to the rest of the whole network, without the need to know the whole network graph. Furthermore, the AML-IP-D provides the logic needed to forward the discovery information between the nodes that requires it. That is, the nodes will only receive other nodes information if they operate under the same topic. As stated in Section 2.3.2, the *eProsima Fast DDS Discovery Server* discovery mechanism implements the desired behaviour of the AML-IP-D.

An example on how the AML-IP nodes will be implemented is shown in figure 13. The AML-IP-D implements a *Fast DDS* Discovery Server, sharing the discovery information of the DDS entities that connect with it.

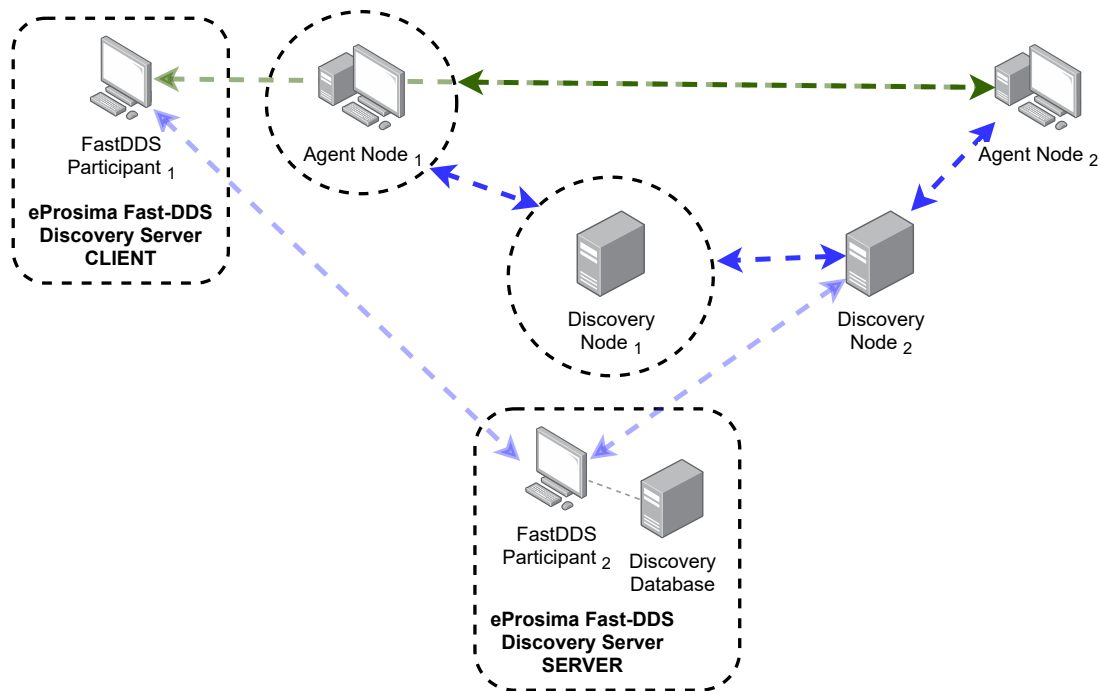


Figure 13: AML-IP Discovery Node design.

3.5 Requirements targeted

This design has been chosen as it fulfills all the requirements and being the most efficient and scalable architecture found. Being aware of the limitations and obstacles this design represents, it is considered by **EPROS** as the way to go for the AML-IP implementation. These are the main challenges encountered when designing the AML-IP, and the potential benefits offered by this architecture, making it a robust and scalable system:

3.5.1 Discovery through WAN

The first challenge is how to manage the dynamic discovery of new AML-IP nodes, which are agnostic to the state of the network to which they are connected. That is, new nodes must discover all other nodes

already in the network with which they can communicate.

The default discovery process in DDS is based on signalling the presence of new DDS entities to all other entities belonging to a multicast group. Thus, the rest of the entities within this multicast group detect the presence of this new entity and start the matching process with it.

Using multicast in large networks is not achievable, and forcing new nodes to be aware of each other beforehand is unfeasible in dynamically growing networks. In such networks, new nodes can arrive and leave at any time, and the other nodes must detect this behaviour.

The architecture of the AML-IP presented here follows the state-of-the-art patterns of P2P networks. It is based on using *well known addresses* to obtain an access point to the entire network. Therefore, the nodes only need to know one point in the network, which will give access to the network status and the address of the rest of the nodes. It is only necessary to have one or more AML-IP-D at *well known addresses* where a new user can connect to gain access to the rest of the discovery information. The AML-IP-D will fulfilled perfectly this feature, as it is explained in Section 3.4.3.

3.5.2 Communication through WAN

The communication through WAN in a P2P mode has important obstacles when working with nodes that are under a NAT connection, which are the most common networks [5]. The main challenges found are the access to an independent point behind a NAT. The AML-IP overcomes these problems and supports the scalability of the AML ecosystem as described in Section 3.4.2.

3.5.3 Node scalability

As the AML project is at a very early stage, it is in a constant state of evolution and may change as the project progresses towards a more stable phase. To overcome this change, a generic AML-IP architecture has been designed for any type of AML node that may be added in the future.

The AML-IP-Alg represents an abstract *AML Node*, referred to as AML-IP-Alg that could be specialized in any kind of node that the future AML requires, as it is described in Section 3.4.1. Joining these nodes with the AML-IP-Agent creates a LAN and WAN AML interconnected ecosystem that could grow dynamically and be easily scalable with new types of nodes and data.

3.6 Benefits and constraints

Some of the limitations that arise in this architecture of the AML-IP are:

- An AML-IP-Agent is needed inside a LAN in order to connect with AML nodes deployed out of the LAN.
- The AML-IP-Agent will centralized the communication with the external networks, being a potential bottleneck in the communication process.

However, this design fulfill every requirement of the project:

- **Decentralized Communication.** The communication is done between AML-IP-Algs deployed in different LANs, i.e. communicating in a WANs. Thus, it is not required any centralized point or service.
- **Publish/Subscribe Communication.** The use of *Fast DDS* as the communication middleware makes it possible to create a data-centric network using the DDS standard.
- **Logical partitions.** Several networks can run simultaneously in the same WAN and be completely independent just by not sharing the network of AML-IP-D.

- **Connection through LAN.** Nodes in the same LAN discover each other by joining a multicast group, and communicate without requiring the deployment of any AML-IP-Agent.
- **Connection through WAN.** AML-IP-Agents in the same WAN discover each other by connecting to an AML-IP-D and communicate to each other P2P.
- **Dynamic discovery.** The AML-IP-D allows the network to scale up or down dynamically and minimises discovery traffic.
- **Data Injection Nodes.** The AML-IP-Alg is an abstract node that can perform any existing or future AML functionality.

Furthermore, the proposed AML-IP architecture has some advantages that exceed the requirements given:

- **Independence between process and communication..** The AML-IP-Agent is not dependent of any AML node and its only purpose is the communication between internal and external nodes. Thus, the AML process will not be affected by possible delays in the communication.
- **Easy Discovery.** It is only needed to know the public IP address and port of an AML-IP-D in order to discover and be discovered by the rest of the network.
- **Computation cluster.** Group of nodes or clusters can work in the same LAN without the need of an AML-IP-Agent, and will discover each other by joining a multicast group. However, these nodes will no be able to communicate with external nodes outside its LAN.

4 Prototype

In order to demonstrate the validity and usability of the design of the AML-IP proposed in this document, an AML-IP Prototype has been developed. It implements the key aspects of the AML-IP that make the prototype a minimum functional version of the final communications framework. This AML-IP prototype is publicly available and is available in the GitHub repository **aml-ip-prototype**¹ under the Open Source Apache 2.0 license ².

4.1 Design

The design to be explained in the this section remains to the first design for the prototype of the AML-IP. This prototype is not aimed to be a robust final tool, but rather a quick-to-implement tool to serve as a Proof of Concept (PoC) for future development of the AML-IP communications framework.

The idea of the prototype is to show a minimal example of DDS communication through WAN with various AML-IP-Alg discovered by one or multiple AML-IP-D. Therefore, this prototype does not implement the AML-IP-Agent due to the complexity and effort it requires.

4.1.1 Prototype Nodes

There are three AML-IP nodes in the prototype to be differentiated:

¹<https://github.com/eProsima/Fast-DDS-monitor>

²<https://github.com/eProsima/Fast-DDS-monitor/blob/main/LICENSE>

- **Proto-Alg-DL:** These nodes will represent every node in the network that publish under a *Description Language* topic. The output of these nodes will be *Proto-Alg-DL relations* that the AML algorithm can understand. This node is the AML-IP-Alg representation of the AML-IP components described in Section 3.4.1.
- **Proto-Alg-Engine:** This node represents the AML node in charge of processing and executing the AML algorithm. It is the node with the highest workload. The input data-stream of this nodes will be the output data of the Proto-Alg-DL nodes. Its output data is the *atomization*, the knowledge built from the data given. This node is the AML-IP-Alg representation of the AML-IP components described in Section 3.4.1.
- **Proto-DiscoveryServer:** This nodes will execute the discovery process that will join the different nodes to those who are related, ergo, to those that must share information that is relevant. It represents an AML-IP-D.

DomainParticipant	DataWriters	DataReaders	Topic	
			Name	Data Type
Proto-Alg-DL	DL		<code>_aml_ip_topic_dloutput</code>	<i>AML_IP_DLOutput</i>
Proto-Alg-Engine	Atom		<code>_aml_ip_topic_atomization</code>	<i>AML_IP_Atomization</i>
		Atom	<code>_aml_ip_topic_atomization</code>	<i>AML_IP_Atomization</i>
		DL	<code>_aml_op_topic_dloutput</code>	<i>AML_IP_DLOutput</i>
Proto-DiscoveryServer				

Table 2: Overview table of user AML-IP node types. The first column shows the name of the node, which corresponds to the name designated for the DomainParticipant. The next two columns specify the DataWriter(s)/DataReader(s) of the DomainParticipant. Finally, the topic of each DataWriter/-DataReader is defined by its name and data type. Since the *AML_IP_DiscoveryServer* node is only involved in the entity discovery process, i.e. it does not apply to the data transfer phase, it does not contain any user DataReader/DataWriter.

4.1.2 Data types

As detailed in Section 2.1, data will be transmitted using *Fast DDS* communications middleware which implements the DDS standard. In a real scenario, the data to be transmitted will depend on the type of the problem on which the AML nodes are working, using different DDS topics to differentiate the type and content of the data. However, for the case of the prototype it is assumed that all AML nodes work on the same ML problem. Even if all nodes work on the same problem, it is required to use different DDS topics defined by the *topic name* and the *topic data type*. In the case of the prototype, the type of data ingested and generated by each type of node will be different. This results in two types of data:

- **DL** (*AML_IP_DLOutput*): Output data from Proto-Alg-DL nodes. This data type contains a sequence of data of type *AML_IP_Relation*. The *AML_IP_Relation* represent the relation of a Term with a number of Constants. It is formed by two sequences of integers of different sizes, and a boolean which specifies the AML relation type.
- **Atomization** (*AML_IP_Atomization*): Output data from Proto-Alg-Engine nodes. It represents the atomization learnt from an execution of the AML algorithm. This data type contains a sequence of data of type *AML_IP_Atom*. The *AML_IP_Atom* data type describes an Atom. It is formed by a sequence of integers of fix sized.

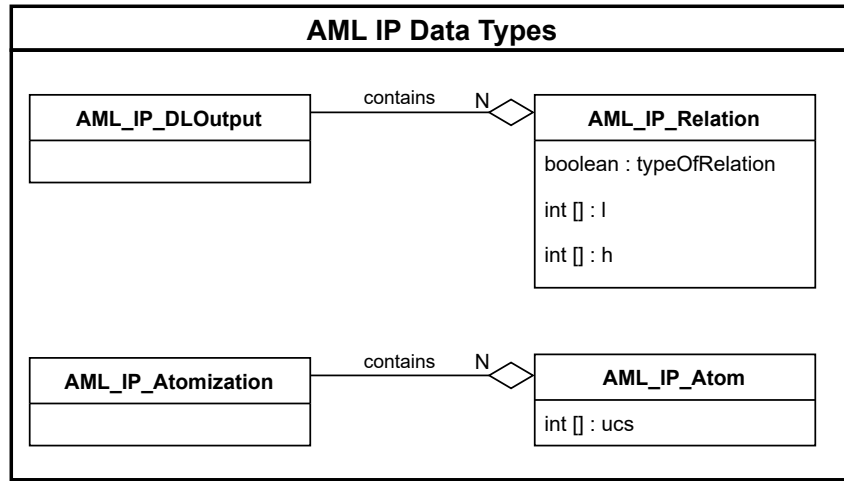


Figure 14: Data Models.

This data types are written in a description language called Interactive Data Language (IDL) [6] that can be read and converted to C++ code by an EPROS tool called *eProxima Fast DDS-Gen*³. The IDL files for the data types that are used in this prototype are shown in Listing 1 and Listing 2. As it could be seen in these snippet codes, every data structure that is shown in 14 is an IDL structure, that will be converted to *std* types in C++.

```

1  // Relation Data Type
2  struct AML_IP_Relation
3  {
4      boolean typeOfRelation;
5      sequence<unsigned long> l;
6      sequence<unsigned long> h;
7  };
8
9  // DL Data Type
10 struct AML_IP_DLOutput
11 {
12     sequence<AML_IP_Relation> relations;
13 };
  
```

Listing 1: Description Language (DL) Data Type IDL description.

```

1  // Atom Data Type
2  struct AML_IP_Atom
3  {
4      sequence<unsigned long> ucs;
5  };
6
7  // Atomization Data Type
  
```

³<https://github.com/eProxima/Fast-DDS-Gen>

```

8  struct AML_IP_Atomization
9  {
10     sequence<AML_IP_Atom> atoms;
11 };

```

Listing 2: Atomization Data Type IDL description.

An example of how a Proto-Alg-Engine node publishes random data that represents an *AML_IP_Atomization* can be found in the source code of the AML prototype publicly available on the GitHub repository `aml-ip-prototype`⁴. In this same repository there is also an example of how the same Proto-Alg-Engine node subscribes to and receives data from a specific topic^{5 6}.

4.2 Experiments

Several experiments has been built and run in order to test the LAN and WAN communication with various network architectures, configurations, and transport methods. This section is focused to explain the experiments execution and discuss the results. These experiments has been divided in two groups: LAN communication and WAN communication.

4.2.1 LAN connectivity

These experiments test the proper behaviour of the prototype with an architecture in which all deployed nodes are on the same LAN. As described in Section 2.3.2, *Fast DDS* supports different transport mechanisms. These are:

- a) User Datagram Protocol (UDP), which is more efficient in LAN communications;
- b) TCP, for WAN communications;
- c) Shared Memory Transport (SHM), as a transport protocol for DDS applications running on the same machine
- d) intraprocess, for DDS entities running in the same system process; and
- e) data-sharing, which makes DataReaders and DataWriters running on the same machine share their database of sent and received data, which in DDS is referred to as *history* of changes.

The experiments performed are as follows:

- **Same host.** Some nodes executing and working in the same host, using UDP, TCP or intraprocess transport.
- **Docker hosts.** Nodes executed in the same physical host but in different logic hosts encapsulated in Docker⁷ containers.

⁴<https://github.com/eProsima/aml-ip-prototype>

⁵Proto-Alg-Engine publication example:
<https://github.com/eProsima/aml-ip-prototype/blob/c6d6de24de7b3201082cb53efba49e01e733c09f/src/cpp/Engine/EngineParticipant.cpp#L240>

⁶Proto-Alg-Engine subscription example:
<https://github.com/eProsima/aml-ip-prototype/blob/c6d6de24de7b3201082cb53efba49e01e733c09f/src/cpp/Engine/EngineParticipant.cpp#L342>

⁷<https://www.docker.com/>

- **Different hosts.** Nodes executed in different machines connected to the same router and under the same NAT.

4.2.2 WAN connectivity

The WAN communication is not a common use case for *Fast DDS*, and therefore exhaustive testing is required in order to check TCP communication in WANs. All the experiments discussed in this section has been tested in a WAN communication.

- **PoC 1**⁸: Test static discovery using TCP connection between Proto-Alg-DL node and Proto-Alg-Engine node.
- **PoC 2**: Test the connection of Proto-Alg-DL and Proto-Alg-Engine nodes, each in different LANs, connected to a AML-IP Prototype Discovery Server Node (Proto-DiscoveryServer) node in an independent LAN.
- **PoC 3**: Test the same connection as PoC 2 but hosting various nodes in the same machine, i.e. one host running an Proto-Alg-DL and an Proto-Alg-Engine nodes, and another machine running an Proto-Alg-Engine and a Proto-DiscoveryServer nodes). This experiment tests: 1. the discovery of AML-IP prototype nodes deployed on different LANs, 2. the discovery of AML-IP prototype nodes deployed on the same LANs, 3. the communication between AML-IP prototype nodes deployed on different LANs, 4. the communication between AML-IP prototype nodes deployed on the same LANs, and 5. the communication between AML-IP prototype nodes deployed on the same host and another in a different LAN.

4.3 Demonstrator

The following section presents a demonstrator of the AML-IP prototype. It details the network architecture deployed for the execution of the prototype, as well as the steps required to download, install and run it. Thus, it is possible to reproduce the results obtained and presented in this document. The main propose of this demonstrator is to illustrate a successful connection between AML-IP prototype nodes deployed along the Internet.

4.3.1 Experiment design

As discussed in Section 4.1, this prototype does not implement the AML-IP-Agent. The nodes implemented are the AML-IP-D and two types of AML-IP-Alg, which are the *Engine* and the *DL*. The AML-IP-Algs connect to a local AML-IP-D that manage the discovery along the Internet but do not forward the data traffic through WAN.

Figure 15 shows the network architecture deployed for this demonstrator. In order to simulate a minimum real scenario, the nodes have been distributed in three different LAN. One of the networks represents a single point of discovery with an Proto-DiscoveryServer, that is a well known address that will distribute the discovery information of every cluster that connects with it. The other two networks are a simulation of real clusters of AML where a single node has been deployed. Each of these networks needs an Proto-DiscoveryServer that knows beforehand the address of the on already in the net. The Proto-Alg-Engine and the Proto-Alg-DL will connect to their respective Proto-DiscoveryServer and will discover each other and communicate directly to each other. This demonstrator does not include the UPnP feature in its *main* version, so the experiment needs a previous phase of port forwarding for every NAT.

⁸This configuration was built in a previous prototype version to allow the connection between specific nodes without a Discovery Server involved. The current version does not allow it, but it could be seen in a previous branch in the repository.

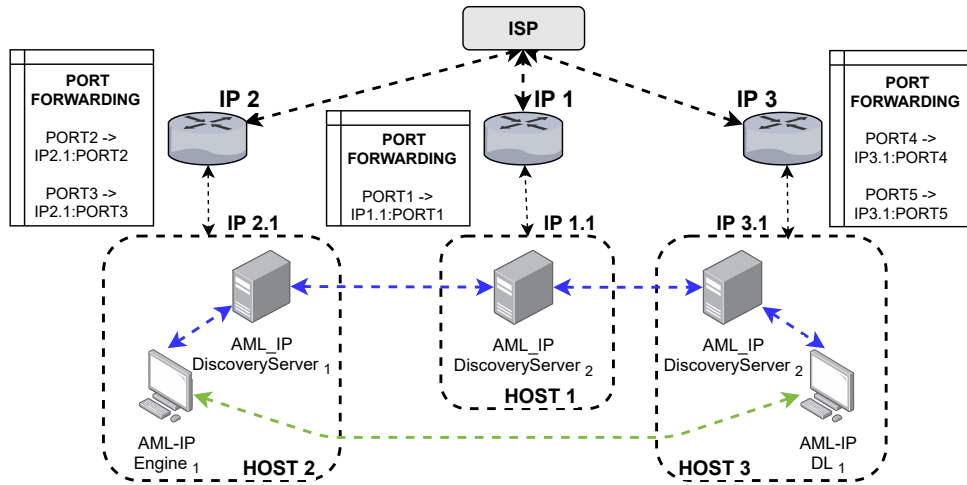


Figure 15: Experiment architecture.

4.3.2 Download and installation

The prototype is been developed as an Open Source project in GitHub. Thus, the whole code and instructions are available for every user in the `aml-ip-prototype` GitHub repository⁹. The project has been developed for Ubuntu 20.04.2.0 LTS (Focal Fossa)¹⁰ Operating System (OS) and it has not been tested in any other platforms.

In the project `README.md` file there are outlined all the instructions to install the libraries and dependencies required to correctly download and compile the project. The main dependencies are the *eProsima Fast DDS* library for DDS communication and the `colcon`¹¹ tool to build the project. It is worth mentioning that it exists a branch in the prototype where a UPnP application is built within the project. However, this branch require to have `Boost 1.71`¹² installed, which is not supported by the `apt`¹³ for the Ubuntu distribution used.

To install the project together with all its dependencies, simply execute the following commands:

```
$ export WORKSPACE_PATH="/path/to/workspace/"
$ mkdir -p ${WORKSPACE_PATH}/src
$ cd ${WORKSPACE_PATH}
$ wget https://raw.githubusercontent.com/eProsima/aml-ip-prototype/main/amlip.repos
$ vcs import src < amlip.repos
$ colcon build
```

⁹<https://github.com/eProsima/aml-ip-prototype>

¹⁰<https://releases.ubuntu.com/20.04/>

¹¹<https://colcon.readthedocs.io/en/released/>

¹²https://www.boost.org/users/history/version_1_71_0.html

¹³<https://wiki.debian.org/Apt>

4.3.3 Experiment setup

For the description of the experiment setup, the terminology of Figure 15 has been followed to refer to the elements involved in the experiment of the prototype. All the executables must be run from the **WORKSPACE** folder and it is required to **source** the installation setup of the project for the prototype node executables to find the required dynamic libraries. For this reason, every command that will be shown from here on will previously execute the following commands in order to correctly configure the environment:

```
$ cd ${WORKSPACE_PATH}
$ source ./install/setup.bash
```

In order to achieve WAN communication between the nodes, it is necessary to perform some port forwarding on the router. This is because the NATs that the routers create do not allow external connections to internal machines if there is not a rule to forward a port to that machine. In order to open these ports manually, the user must enter in the router configuration and create a new rule to forward a specific port to the host desired, as it is shown in Figure 16.

The following describes the configuration and the steps taken to set up the machines running the prototype.

• HOST 1

This host executes a single Proto-DiscoveryServer that listens in its public address and the port forwarded in the router. First, the administrator of **HOST 1** must configure its router to set a static IP to its machine, which is **IP1.1** according to Figure 15. Then, it set a port forwarding from the router to its machine. As a result, every external request to address **IP1:PORT1** will be forwarded to **IP1.1:PORT1** and will be received by the Proto-DiscoveryServer.

The following command has been used in order to run this node

```
$ ./install/amlip/bin/AML_IP_DiscoveryServer \
    --id=1 \
    --listening-addresses="${IP1},${PORT1}" \
    --time=0
```

Listing 1: Bash command to run the Proto-DiscoveryServer node in **HOST 1**.

where

- **id** sets this Proto-DiscoveryServer id to 1, information required to connect with it;
- **listening-addresses** sets the address or addresses where the Proto-DiscoveryServer is listening; and
- **time** set to 0 implies that the Proto-DiscoveryServer will be running until the **Enter** key is pressed.

• HOST 2

This host executes an Proto-DiscoveryServer that communicates with the Proto-DiscoveryServer deployed in **HOST 1** and listens in **localhost** in order to discover every node working on **HOST 2**. It will also execute an Proto-Alg-Engine that will listen to every Proto-Alg-DL node publishing DL data under the **AML_IP_DLOutput** topic and to every Proto-Alg-Engine publishing Atomization data under the **AML_IP_Atomization**. In turn, this Proto-Alg-Engine node will publish its own Atomization information under the latter topic at constant time intervals and under the data type format according to Figure 14.

First, as done in **HOST 1**, the administrator of **HOST 2** must configure its router to set a static IP to its machine, which is **IP2.1** according to Figure 15. Then, it set two ports forwarding from the router to its machine. Thus, every external request to address **IP2:PORT2** will be forwarded to **IP2.1:PORT2** and every external request to address **IP2:PORT3** forwarded to **IP2.1:PORT3**.

Secondly, the Proto-DiscoveryServer is executed with the following command:

```
$ ./install/amlip/bin/AML_IP_DiscoveryServer \
  --id=2 \
  --listening-addresses="${IP2},${PORT2};127.0.0.1,5000" \
  --connection-addresses="${IP1},${PORT1},1" \
  --time=0
```

Listing 2: Bash command to run the Proto-DiscoveryServer node in **HOST 2**.

where

- **id** sets this Proto-DiscoveryServer id to 2, which is required to connect with it;
- **listening-addresses** is set to listen in the port forwarded in the router in order to receive the information from the **HOST1**, and to the **localhost=127.0.0.1** address to communicate with any node internal to **HOST 2**;
- **connection-addresses** is the remote Proto-DiscoveryServer with which this Proto-DiscoveryServer is going to connect; and
- **time** set to 0 implies that the Proto-DiscoveryServer will be running until the **Enter** key is pressed.

Finally, the **HOST 2** administrator runs the Proto-Alg-Engine with the following command:

```
$ ./install/amlip/bin/AML_IP_Engine \
  --listening-addresses="${IP2},${PORT3};127.0.0.1,4000" \
  --connection-addresses="127.0.0.1,5000,2" \
  --period 4 \
  --size 100 \
  --samples 0
```

Listing 3: Bash command to run the Proto-Alg-Engine node in **HOST 2**.

where

- **listening-addresses** sets the addresses to which the Proto-Alg-Engine node will listen, i.e. the channel on which it intends to receive information from the topics to which its DataReaders subscribe;
- **connection-addresses** sets the addresses of the Proto-DiscoveryServer deployed in the same machine;
- **period** indicates the interval in seconds to publish samples;
- **size** sets the size of the internal message; and
- **samples** sets the number of samples to publish.

For the Proto-Alg-Engine deployed on **HOST 2**, the **listening-addresses** has two values. The first one corresponds to the address through which the machine is accessed remotely, in this case this

will be a WAN communication but it is also valid for LAN communications as long as the router supports loopback to the internal network, while the second one corresponds to the communications on the same machine.

• HOST 3

This host presents a similar configuration than the **HOST 2** but running an Proto-Alg-DL instead of an Proto-Alg-Engine. First, the administrator of **HOST 3** must configure its router to set a static IP to its machine, which is **IP3.1** according to Figure 15. In this case, there is no need to forward any port since the TCP communication is able to use an already opened connection in order to receive and send data. Thus, only one of the nodes in the communication requires to open an access from the router, while the other can relay in the TCP connection already opened.

Secondly, the Proto-DiscoveryServer is executed with the following command, similar to the one explained for **HOST 2** but changing the ip and id configuration options.

```
$ ./install/amlip/bin/AML_IP_DiscoveryServer \
    --id=3 \
    --listening-addresses="${IP3},${PORT4};127.0.0.1,5002" \
    --connection-addresses="${IP1},${PORT1},1" \
    --time=0
```

Listing 4: Bash command to run the Proto-DiscoveryServer node in **HOST 3**.

Finally, the **HOST 3** administrator run the Proto-Alg-DL with the following command, similar to the one used for running the Proto-Alg-Engine in **HOST 2**.

```
$ ./install/amlip/bin/AML_IP_DL \
    --id=3 \
    --listening-addresses="${IP3},5001;127.0.0.1,5003" \
    --connection-addresses="127.0.0.1,5002,3" \
    --period 2 \
    --size 50 \
    --samples 0
```

Listing 5: Bash command to run the Proto-Alg-DL node in **HOST 3**.

4.3.4 Results

This experiment has been executed using the prototype implemented and following the instructions already mentioned. The experiment has been executed remotely with each host deployed in a personal network. In the Appendix A could be seen different screenshots taken after the execution explained in Section 4.3.3.

Figure 17 represents the execution of the first Proto-DiscoveryServer in **HOST 1** after executing 1. In this screenshot could be seen the different Proto-DiscoveryServers that connect to it and the discovery information of the nodes. This information is reported by all the Proto-Alg-DL and distributed to other Proto-Alg-DLs. Finally the two remote Proto-DiscoveryServers leave the network, and therefore the discovery information from the working nodes is no longer distributed. This means that the nodes that already know each other could still communicate to each other, but any other new node will not discover them, nor will currently connected nodes have a periodic report on the status of remote nodes.

Figure 18 shows how the Proto-DiscoveryServer in **HOST2** connects and discovers the central Proto-DiscoveryServer deployed on **HOST1** after running the command 2. Once the local machine is already

connected with the network of Proto-DiscoveryServers, a new Proto-Alg-Engine is instantiated by running the command 3 in **HOST 2**.

Finally, in **HOST 3** the same deployment configuration is performed by executing an Proto-DiscoveryServer with command 4 and a Proto-Alg-DL with command 5. It could be seen the execution of this Proto-DiscoveryServer in **HOST 3** in Figure 20.

Once everything is deployed, the two nodes will discover each other by the network of Proto-DiscoveryServers and will start sending and receiving data. This could be seen in Screenshot 21, which shows the data being sent by the Proto-Alg-DL. In the Screenshot 19 it can be seen how the Proto-Alg-Engine publish its own kind of data, but since there are no subscribed nodes, none receives it. Furthermore, it could be seen how this Proto-Alg-Engine receives data from a remote Proto-Alg-DL, which is in this case in **HOST 3**.

The following details how to understand the output of an Proto-Alg-Engine and Proto-DiscoveryServer execution in the prototype (Figures 21 and 19). The output messages clearly show the two phases that are specified in Section 3.3.1:

- Discovery phase: the first output messages refer to the discovery of nodes in the network.
- Data transfer phase: both the Proto-Alg-Engine node and the Proto-Alg-DL node start publishing data. There are two kinds of messages.
 - Those that start with << refer to messages **published** by the node. The following output message taken from these executions shows how the Proto-Alg-DL send a message in topic *AML-IP-DLOutput* with 8 relations.

```
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 12
message: < Number of relations: 8 >
```

- Those that start with >> refer to messages **received** from other nodes.

Then the remote Proto-Alg-Engine receive this same message with the same number of relations in topic *AML-IP-DLOutput* to which it has been subscribed beforehand. This is represented in the following output message.

```
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <
Number of relations: 8 > from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 Number of
messages received so far: 12
```

4.3.5 Experiment Conclusions

The AML-IP prototype has been used to test the feasibility and viability of the final AML-IP design. From the experiment carried out with this prototype, the following conclusions can be drawn about the proposed final design:

- To achieve dynamic discovery and communication between nodes in WAN, only one central node is required, which performs the discovery service and to which the rest of the nodes of the network will be connected.
- To achieve WAN node communication, i.e. nodes deployed in different LANs under a NAT, it is necessary to apply a network configuration, which can be either manual, performed by the user on his private network router, or automatic through a tool that implements the UPnP protocol.
- The fact of having two different phases for communication, i.e. the discovery phase and the data transfer phase, makes the discovery of new entities joining the network independent from the communication already established between stable entities in the network.

5 Conclusions

This document described the first approach for the design of the AML-IP. It is not intended to be the final design but a first sketch on what it going to be built, and it could change along the project depending on the necessities that appear or the limitations encountered during the development process. However, it is intended to be a complete, robust and functional design fulfilling all the ALMA project requirements. The main advantages of this design are described below:

5.1 Advantages

The AML-IP communication platform described in this document address all requirements set for a correct and efficient execution of an AML environment.

Reliable software tools This architecture remains over different already implemented and tested products, which facilitates the implementation and reduces the errors that could be found. Moreover, all of these products are released under the Apache 2.0 Open Source license, and are under support periods by *EPROS*.

The development of the communication platform would be focus on the AML environment, and so it will count with specific improvements in usability and efficiency as stated in the ALMA Project GA.

User centered design The design of the AML-IP is a User Centred Design, i.e. the entire user experience with the communications framework has been taken into consideration. Thus, AML users could make use of the AML-IP communication benefits without the need of know, configure or understand the communication protocols.

The independence between the AML-IP and the AML nodes is very useful for the AML final user. The only requirement for this user in order to use the AML-IP will be to execute an internal AML-IP-Agent in its LAN in case it wants to communicate over WAN, and to know the network address, i.e. IP-port pair, of a remote AML-IP-D. This will be addressed in future versions by maintaining constant points of discovery over the Internet.

Network Scalability The architecture proposed for the AML-IP is highly scalable. Every AML-IP-Alg is independent of the network state and the number of remote nodes connected. Moreover, the AML-IP framework is also aware of the nodes in the same LAN. This allows to build robust local clusters independent of other components deployed outside the user's private network.

Finally, the discovery mechanisms used for a dynamic AML-IP node discovery over WAN are highly reliable due to the distribution of the discovery information along redundant networks. Furthermore, the information sent along WAN connections between AML-IP-Agent is only forwarded to the nodes that requires this information, i.e. the data is only transmitted to the nodes subscribed to the same topics under which the data has been published. This highly reduces the network traffic against other P2P possible solutions as Block Chain.

Security The communications security when working in the Internet is crucial, so it is a matter to be taken into account in a communications framework such as AML-IP. The AML-IP separates the data required to train an AML algorithm, and the data of the algorithm itself, avoiding to share personal or private information.

The user authentication, access permissions and the encryption of the communication over the Internet is given by *Fast DDS* communications middleware, avoiding extra security features. There is the possibility to increase the point to point communication security by using TLS over TCP, but it may not be a desired

feature as the AML-IP is design to be an open environment where any new node could enter the network and receive the whole information. The possibility of adding a logging or authentication mechanism to enter a network of AML-IP-D nodes will be considered for future versions of the AML-IP.

5.2 Limitations

There are some limitations in the design presented here. These have been considered and taken into account in order to be addressed in future versions of the AML-IP. The constraints identified, together with the potential approaches proposed to overcome them, are presented below.

Centralize communication from LAN to WAN The AML-IP-Agent centralize the connection between the user private network and the Internet, while creates a secure, independent WAN communication that do not affect to the correct operation of the AML nodes. Therefore, it is a single point of failure in the AML environment. However, the communication over WAN of nodes behind a NAT is a well-known problem with very limited options to address it.

The approach studied to overcome this issue is to allow multiple AML-IP-Agent nodes inside a LAN, what would avoid the single point of failure by deploying redundant nodes. Nevertheless, this idea must be further discussed, as this feature has to be carefully designed in order to avoid sending duplicated data and to manage the UPnP protocol.

Large amount of data AML-IP-Alg nodes are expected to generate and transmit a large amount of data. This is a problem for communication proposes, as the whole data is needed to be sent from one AML-IP-Alg to the rest of the network in a reliable way and with a higher frequency than new data is generated.

The first way considered to address this problem is to fragment the data to be sent reliably. Another approach considered is to reduce the amount of data required by an AML node. Thus, not all the data generated by an AML node would be sent again, but only the update of old data. The possible features of this functionality will be discussed and agreed with ALG and the AML development team.

A Screenshots of Demonstrator results

network

DHCP	NAT/PAT	DNS	UPnP	DynDNS	DMZ	NTP	ONT
------	---------	-----	------	--------	-----	-----	-----

These rules are needed to authorise a remote communication from Internet to reach a specific device of your LAN. You can also define the port(s) that this communication will use.



warning : make sure you have not filtered these ports in the firewall.

customize rules

status	application / service	internal port	external port	protocol	device	activate	
	Web Server (HTTP) ▼	80	80	TCP ▼	HUAWEI ▼		save
✓	AML-IP Discovery Server 1	6000	6000	TCP	Laptop eProxima	✓	delete
✓	AML-IP Engine	7000	7000	TCP	Laptop eProxima	✓	delete

Figure 16: Example of port forwarding configuration on a private user router.

```
eProxima@eProxima-XP5-15-7590:~/AML$ ./install/amlip/bin/AML_IP_DiscoveryServer --id=1 --listening-addresses="37.11.18.30,5000" --time 0
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43
with id 1 and GUID 01.0f.01.41.4c.47.45.42.52.41.49.43
listening in addresses:
IP: 37.11.18.30 port: 5000
connecting with servers in addresses:
Please press enter to stop it at any time.
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43 discovered participant 01.0f.02.41.4c.47.45.42.52.41.49.43
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43 discovered participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43 discovered publisher 01.0f.47.a1.f3.16.0f.8d.01.00.00.00
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43 discovered subscriber 01.0f.47.a1.f3.16.0f.8d.01.00.00.00
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43 discovered subscriber 01.0f.47.a1.f3.16.0f.8d.01.00.00.00
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43 discovered participant 01.0f.03.41.4c.47.45.42.52.41.49.43
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43 discovered participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43 discovered publisher 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43 removed participant 01.0f.02.41.4c.47.45.42.52.41.49.43
DiscoveryServer Participant 01.0f.01.41.4c.47.45.42.52.41.49.43 removed participant 01.0f.03.41.4c.47.45.42.52.41.49.43
eProxima@eProxima-XP5-15-7590:~/AML$
```

Figure 17: Proto-DiscoveryServer in HOST 1

```

~/aml-ip-prototype ws$
$ ./install/amlip/bin/AML IP DiscoveryServer --id=2 --listening-addresses="127.0.0.1,5000;87.219.112.233,6000" --connection-addresses="37.11.18.30,5000,1" --time=0
DiscoveryServer Participant 01.0f.02.41.4c.47.45.42.52.41.49.43
with id 2 and GUID 01.0f.02.41.4c.47.45.42.52.41.49.43
listening in addresses:
IP: 127.0.0.1 port: 5000
IP: 87.219.112.233 port: 6000
connecting with servers in addresses:
IP: 37.11.18.30 port: 5000 id: 1
Please press enter to stop it at any time.
DiscoveryServer Participant 01.0f.02.41.4c.47.45.42.52.41.49.43 discovered participant 01.0f.01.41.4c.47.45.42.52.41.49.43
DiscoveryServer Participant 01.0f.02.41.4c.47.45.42.52.41.49.43 discovered participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00
DiscoveryServer Participant 01.0f.02.41.4c.47.45.42.52.41.49.43 discovered publisher 01.0f.47.a1.f3.16.0f.8d.01.00.00.00
DiscoveryServer Participant 01.0f.02.41.4c.47.45.42.52.41.49.43 discovered subscriber 01.0f.47.a1.f3.16.0f.8d.01.00.00.00
DiscoveryServer Participant 01.0f.02.41.4c.47.45.42.52.41.49.43 discovered subscriber 01.0f.47.a1.f3.16.0f.8d.01.00.00.00
DiscoveryServer Participant 01.0f.02.41.4c.47.45.42.52.41.49.43 discovered participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
DiscoveryServer Participant 01.0f.02.41.4c.47.45.42.52.41.49.43 discovered publisher 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00

~/aml-ip-prototype ws$

```

Figure 18: Proto-DiscoveryServer in HOST 2.

```

~/aml-ip-prototype ws$
$ ./install/amlip/bin/AML IP Engine --listening-addresses="87.219.112.233,7000;127.0.0.1,4000" --connection-addresses="127.0.0.1,5000,2" --period 4 --size 100 --samples 0
Engine Participant created with guid: 01.0f.47.a1.f3.16.0f.8d.01.00.00.00
listening in addresses:
IP: 87.219.112.233 port: 7000
IP: 127.0.0.1 port: 4000
connecting with servers in addresses:
IP: 127.0.0.1 port: 5000 id: 2
Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 publishing. Please press enter to stop it at any time.

<< Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 sent atomization number: 1 message: <Number of atoms: 41>
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 sent atomization number: 2 message: <Number of atoms: 10>
Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 matched DL reader with a new DL: 1.f.3a.1f.d7.d5.c7.1d.1.0.0.0.0.1.3
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 3> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 1

<< Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 sent atomization number: 3 message: <Number of atoms: 74>
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 15> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 2
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 34> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 3

<< Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 sent atomization number: 4 message: <Number of atoms: 54>
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 11> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 4
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 43> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 5

<< Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 sent atomization number: 5 message: <Number of atoms: 84>
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 32> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 6
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 32> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 7

<< Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 sent atomization number: 6 message: <Number of atoms: 19>
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 48> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 8
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 13> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 9

<< Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 sent atomization number: 7 message: <Number of atoms: 41>
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 35> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 10
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 25> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 11

<< Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 sent atomization number: 8 message: <Number of atoms: 89>
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 8> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 12
>> Engine Participant 01.0f.47.a1.f3.16.0f.8d.01.00.00.00 receive DL message <Number of relations: 10> from: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
. Number of messages received so far: 13
~/aml-ip-prototype ws$

```

Figure 19: Proto-Alg-Engine in HOST 2.

```

AML_IP_DiscoveryServer
paris@paris-XPS-15-9570:~/projects/AML-IP$ ./install/amlip/bin/AML_IP_DiscoveryServer --listening-addresses="127.0.0.1,5002;37.35.177.91,5008" --connection-addresses="37.11.10.30,5000,1" --time 8 --id 3
DiscoveryServer Participant 01.0f.03.41.4c.47.45.42.52.41.49.43
with id 3 and GUID 01.0f.03.41.4c.47.45.42.52.41.49.43
listening in addresses:
IP: 127.0.0.1 port: 5002
IP: 37.35.177.91 port: 5008
connecting with servers in addresses:
IP: 37.11.10.30 ports 5000 id: 1
Please press enter to stop it at any time.
DiscoveryServer Participant 01.0f.03.41.4c.47.45.42.52.41.49.43 discovered participant 01.0f.01.41.4c.47.45.42.52.41.49.43
DiscoveryServer Participant 01.0f.03.41.4c.47.45.42.52.41.49.43 discovered participant 01.0f.47.a1.f3.16.0f.0d.01.00.00.00
DiscoveryServer Participant 01.0f.03.41.4c.47.45.42.52.41.49.43 discovered publisher 01.0f.47.a1.f3.16.0f.0d.01.00.00.00
DiscoveryServer Participant 01.0f.03.41.4c.47.45.42.52.41.49.43 discovered subscriber 01.0f.47.a1.f3.16.0f.0d.01.00.00.00
DiscoveryServer Participant 01.0f.03.41.4c.47.45.42.52.41.49.43 discovered subscriber 01.0f.47.a1.f3.16.0f.0d.01.00.00.00
DiscoveryServer Participant 01.0f.03.41.4c.47.45.42.52.41.49.43 discovered participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
DiscoveryServer Participant 01.0f.03.41.4c.47.45.42.52.41.49.43 discovered publisher 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
paris@paris-XPS-15-9570:~/projects/AML-IP$

```

Figure 20: Proto-DiscoveryServer in HOST 3.

```

amlip DL
paris@paris-XPS-15-9570:~/projects/AML-IP$ ./install/amlip/bin/AML_IP_DL --connection-addresses="127.0.0.1,5002,3" --listening-addresses="37.35.177.91,5001;127.0.0.1,5003" --period 2 --size 50 --samples
0
DL Participant created with guid: 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00
listening in addresses:
IP: 37.35.177.91 port: 5001
IP: 127.0.0.1 port: 5003
connecting with servers in addresses:
IP: 127.0.0.1 port: 5002 id: 3
DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 publishing. Please press enter to stop it at any time.
DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 matched with 1.f.47.a1.f3.16.f.0d.1.0.0.0.0.2.4
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 1 message: <Number of relations: 3>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 2 message: <Number of relations: 15>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 3 message: <Number of relations: 34>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 4 message: <Number of relations: 11>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 5 message: <Number of relations: 43>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 6 message: <Number of relations: 32>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 7 message: <Number of relations: 32>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 8 message: <Number of relations: 48>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 9 message: <Number of relations: 13>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 10 message: <Number of relations: 35>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 11 message: <Number of relations: 25>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 12 message: <Number of relations: 8>
<< DL Participant 01.0f.3a.1f.d7.d5.c7.1d.01.00.00.00 sent DLOutput number: 13 message: <Number of relations: 18>
paris@paris-XPS-15-9570:~/projects/AML-IP$

```

Figure 21: Proto-Alg-DL in HOST 3.

References

- [1] *eProxima Fast DDS Documentation - What is DDS?* Version 2.3.1. Proyectos y Sistemas de Mantenimiento SL (eProxima), 2021. URL: https://fast-dds.docs.eprosima.com/en/v2.3.1/fastdds/getting_started/definitions.html.
- [2] *eProxima Fast DDS Documentation*. Version 2.3.1. eProxima, 2021. URL: <http://fast-dds.docs.eprosima.com/en/v2.3.1/>.
- [3] *eProxima Fast DDS Documentation - What is DDS?* Version 2.3.1. Proyectos y Sistemas de Mantenimiento SL (eProxima), 2021. URL: https://fast-dds.docs.eprosima.com/en/v2.3.1/fastdds/getting_started/definitions.html.
- [4] *IANA Guidelines for IPv4 Multicast Address Assignments*. RFC-3171. Internet Engineering Task Force, 2001. URL: <http://datatracker.ietf.org/doc/html/rfc3171>.
- [5] Zhou Hu. *Abstract NAT Traversal Techniques and Peer-to-Peer Applications*.
- [6] *Interface Definition Language*. Version 4.2. Object Management Group, Mar. 2018. URL: <https://www.omg.org/spec/IDL/4.2/PDF>.