



Human-centric algebraic machine learning

AML-IP implementation - Revised M18

Deliverable D6.2

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: **D6.2**
 Title: **AML-IP implementation - Revised M18**
 Month: 18
 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: 2022-02-28
 Version: v3.4
 Status: Finished

Version history

Version	Date	Responsible	Author/Reviewer	Comments
v1.0	2021.05.04	EPROS	Javier París Uhryn	D6.1 - First Draft
v2.0	2021.05.18	EPROS	Raúl Sánchez-Mateos Lizano	D6.1 - Second Draft
v2.1	2021.05.28	EPROS	Raúl Sánchez-Mateos Lizano	D6.1 - Finished document
v3.0	2022.02.07	EPROS	Javier París Uhryn	D6.2 - First Draft
v3.1	2022.02.22	EPROS	Juan López Fernández	D6.2 - First Review (RV1)
v3.2	2022.02.23	EPROS	Javier París Uhryn	D6.2 - Apply changes from RV1
v3.3	2022.02.24	EPROS	Juan López Fernández	D6.2 - Second Review (RV2)
v3.4	2022.02.28	EPROS	Javier París Uhryn	D6.2 - 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 D6.2 - AML-IP implementation - Revised M18* of the ALMA project. This deliverable includes the first implementation of the AML Integrating Platform (AML-IP), using nodes that mock AML ones for data generation, and establishing communication via Robotic Operative System v2 (ROS 2) nodes. The goal is to develop an easy-to-deploy platform for cloud and edge computing environments exploiting the parallelization benefits of AML and the integration facilities of a robotic framework as ROS 2. The platform interconnects AML components (such as engines) and hooks these with other software including data injection nodes, task managers and edge nodes.

Revision update

Revision of deliverable *D6.1 - AML-IP implementation - Initial* for *Work Package 6: System and Tools*.

Major Changes

- **AML-IP Agent Node (AML-IP-Agent) underneath technology:** Previous deliverable stated that Integration Service (IS) would be the base platform on which to implement the AML-IP-Agent. After thorough research and elaboration of proofs of concept with IS, it has been evinced that this is not the most appropriate tool for the purpose at hand. eProxima DDS Router (DDS Router), a novel application designed to fulfill all requirements specific to the ALMA project, is to be used instead.
- **AML-IP scenarios divisions:** Deliverable *D6.1 - AML-IP implementation - Initial* focused on exploiting the **collaborative** capabilities of AML, by implementing a prototype for distributing models among users working on the same problem. In this version a new working scenario has been added, in which the main task is to **distribute** the **workload** among multiple nodes offering their computational resources.

Progress

- **AML-IP-Agent implementation (3.4.2):** The AML-IP-Agent will relay on the DDS Router for its main behaviour, this product being already released and currently in the beta-testing phase.

New modules

- **AML-IP distributed workload scenario (5):** The AML-IP has evolved in distributed workload scenarios. This is a scenario for the AML-IP to exploit the AML features that allow for the parallelization of the training process between independent computing nodes. Within this new scenario, two new kind of nodes are presented: AML-IP Algebraic Main Node (AML-IP-Main) and AML-IP Algebraic Computing Node (AML-IP-Computing).
- **AML-IP distributed workload scenario prototype (6):** There is a new prototype of the AML-IP for *workload distribution scenarios* that, along with a demonstrator, illustrates the new functionality of the above-mentioned nodes.

Contents

Executive Summary	3
Revision Update	4
Contents	7
Acronyms	8
1 Introduction	9
1.1 Purpose of document	9
1.2 Document sections	9
1.3 Partners involved	10
2 AML Environment Overview	10
2.1 Node oriented design	10
2.2 AML-IP scenarios	11
2.2.1 Collaborative learning scenario	11
2.2.2 Workload distribution scenario	11
2.3 Communication between AML nodes	11
2.4 Requirements	12
2.5 Enabling technologies	13
2.5.1 Data Distribution Service (DDS)	13
2.5.2 eProsima Fast DDS	14
2.5.3 Fast DDS Python Bindings	15
2.5.4 miniUPnP	15
3 AML-IP Architecture	15
3.1 AML-IP architecture description	16
3.2 Use Cases	17
3.3 AML-IP functional operation	18
3.3.1 Phases of communication	18
3.3.2 AML-IP operational area networks	19
3.3.3 AML-IP Communication in LAN	19
3.3.4 AML-IP Communication in WAN	20
3.3.5 AML-IP Communication Sequence Diagram	23
3.4 AML-IP Components	24
3.4.1 AML-IP Algebraic Node	24
3.4.2 AML-IP Agent	24
3.4.3 AML-IP Discovery Server	30
3.5 Requirements targeted	30
3.5.1 Discovery through WAN	30
3.5.2 Communication through WAN	31
3.5.3 Node scalability	31
3.6 Benefits and constraints	31
4 AML-IP architecture scenario Prototype	32
4.1 Design	32
4.1.1 Prototype Nodes	32
4.1.2 Data types	33
4.2 Experiments	35

4.2.1	LAN connectivity	35
4.2.2	WAN connectivity	36
4.3	Demonstrator	36
4.3.1	Experiment design	36
4.3.2	Download and installation	37
4.3.3	Experiment setup	38
4.3.4	Results	40
4.3.5	Experiment Conclusions	41
5	AML-IP for workload distribution scenario Design	41
5.1	Workload distribution scenario	42
5.2	AML-IP for workload distribution scenarios	42
5.2.1	Architecture Description	42
5.2.2	Communication	43
5.2.3	Synchronization	44
5.2.4	Multiple scenarios	44
5.3	MultiService Communication Protocol	45
5.3.1	Description	45
5.3.2	Requirements	46
5.3.3	Design	47
5.3.4	Advantages and Limitations	49
5.3.5	Topics and data types	50
5.3.6	Use cases	51
5.3.7	Alternatives	52
6	AML-IP workload distribution scenario Prototype	52
6.1	Technology used	53
6.1.1	AML-IP Algebraic Nodes	53
6.1.2	Fast DDS Python Bindings	53
6.1.3	eProsima DDS Router	53
6.2	Design	53
6.2.1	Prototype nodes	53
6.2.2	DDS Data types	54
6.3	Experiments	55
6.3.1	Simple experiment	55
6.3.2	WAN experiment	55
6.3.3	Complex experiment	55
6.4	Demonstrator	56
6.4.1	Experiment design	56
6.4.2	Download and installation	56
6.4.3	Experiment setup	57
6.4.4	Results	57
6.4.5	Experiment Conclusions	58
7	Conclusions	58
7.1	Advantages	59
7.2	Limitations	60
A	Screenshots of Demonstrator results	61
B	IDLs for workload distribution scenario prototype	64

C Screenshots of Demonstrator for workload distribution results	66
References	69

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-Computing	AML-IP Algebraic Computing Node.
AML-IP-D	AML-IP Discovery Node.
AML-IP-DL	AML-IP Algebraic Description Language Node.
AML-IP-Engine	AML-IP Algebraic Engine Node.
AML-IP-Main	AML-IP Algebraic Main Node.
DDS	Data Distribution Service.
DDS Router	eProsima DDS Router.
DL	Description Language.
GA	Grant Agreement.
IDL	Interactive Data Language.
IGD	Internal Gateway Device.
IP	Integrating Platform.
IS	Integration Service.
LAN	Local Area Network.
ML	Machine Learning.
MSCP	MultiService Communication Protocol.
NAT	Network Address Translation.
OS	Operating System.
P2P	Peer to Peer.
PoC	Proof of Concept.
QoS	Quality of Service.
ROS 2	Robotic Operative System v2.
RPC	Remote Procedure Call.
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 approach to Machine Learning (ML) known as AML. AML is a novel 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.

The AML-IP is a framework that allows for the simple deployment of an AML distributed architecture. This design has been elaborated 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*.

The AML-IP is the cornerstone by which the rest of the project components are underpinned. It will serve as a platform for non-expert users to create and manage different AML nodes and the communication between them, in local and remote networks. It will also allow to create complex distributed networks, with one or multiple users working collaboratively in the same problem, and provide an easy to use framework to facilitate the process of different aspects of the AML process, as parallelizing tasks, sharing models, etc.

1.1 Purpose of document

This document describes in detail many aspects of the AML-IP; its scope, purpose, design, and the current state of its implementation and deployment. This is a complex platform that must support an evolving project. Therefore, the decisions made so far, as well as the designs and diagrams may be modified depending on the necessities of the project.

Two different AML-IP prototypes are presented in this document, and each of them refer to different components of the final AML-IP. These parts are not mutually-exclusive, but the corresponding prototypes are independent to one another. Along with these prototypes, instructions on how to execute a demo are provided, as well as a demonstrator for each of them showcasing the attained results.

This document is a revision of deliverable *D6.1 - AML-IP implementation - Initial*. It incorporates new aspects of the project, eliminates deprecated information, and updates those parts of the AML-IP design that have suffered modifications over the last months of development. Main changes and additions to this revision not present in the initial manuscript are listed in Revision update.

1.2 Document sections

- **AML Environment Overview** (2) This section summarizes the AML environment to present the motivation for the development of the AML-IP.
- **AML-IP Architecture** (3) Here, the AML-IP requirements are presented, as well as the design chosen and the decisions made in this process.
- **AML-IP collaborative learning scenario Prototype** (4) The first prototype refers to the collaborative learning scenario of a future AML network, and adds designs, images and links to the prototype implementation, as instructions on how to execute it.
- **AML-IP for workload distribution scenario** (5) In this section a specific AML scenario is described, and how the AML-IP will provide with the needed architecture to communicate every node.
- **AML-IP workload distribution scenario Prototype** (6) The second prototype simulates the workload distribution scenario and presents the design and implementation draft, along with a

demonstrator and the instructions on how to execute it.

1.3 Partners involved

Team Leader:

- Proyectos y Sistemas de Mantenimiento SL (eProsima) (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 accommodating the distinct AML components (such as AML-IP Algebraic Engine Node (AML-IP-Engine) nodes, AML-IP-Computing nodes or any other future component), and managing the intercommunication between these over the Internet.

2.1 Node oriented design

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

A node must be understood as a piece of software that runs in a specific machine at a specific time. An AML network is made up of one or multiple nodes communicating with each other. A node has the following characteristics:

- It is independent of the rest of the network. This means that any node could work on its own, or in any network configuration, being able to manage its own communication and status. Nonetheless, some specific nodes could alter or modify the behaviour of others if they are designed to do so.
- The communication is not limited to the virtual or physical environment of the node. Any node has the potential to communicate with any other as long as they are connected to the same network (e.g. the Internet).
- Each kind of node has its own expected behaviour, and all nodes belonging to the same kind display exactly the same behaviour (when configured with the same settings).
- There is no limit in the amount of nodes (or the amount of nodes of the same kind) in a network. Nevertheless, some groups of nodes could make no sense without others, as many tasks require some kinds of nodes to be processed.

A node-based environment for the AML-IP offers many advantages. These are the main ones:

- **Independence:** Each node runs in a process, and is independent of the rest of the nodes. The data this node uses, the communication interfaces and its status and execution is managed inside the node, making it possible for any node to work alone and independently of the environment it is running. Obviously, it would not make any sense to run some nodes alone. But still, the process would behave as expected in this or any other scenarios.
- **Adaptability:** As any node is independent from the rest, the size and form of these networks is limitless. Any amount of nodes could work at the same time, only being affected by external factors (as network failures or saturation).

- **Dynamic deployment:** As any node is independent, the deployment of a network is straightforward given that any node will run on its own and no synchronization is needed. Also, to change the behaviour of the whole network, it would be enough to dispose some nodes and raise others, without affecting the rest of the network.
- **Scalability:** Changing the behaviour of a kind of node is easy, as all of them share the same code and are independent from the rest. Also, by creating new kinds of nodes many new configurations may be conceived, making the whole system flexible and growth-friendly.
- **Maintainability:** A large part of the node implementation is generic for all of them. In addition, each kind of node shares the same implementation. Therefore, the amount of code to implement and maintain is greatly reduced, and highly reusable.

2.2 AML-IP scenarios

One of the biggest strengths of AML is the great parallelization capabilities that its design enables, allowing different kind of nodes to share their data in order to distribute the computational effort. These parallelization possibilities can be divided in two main types, depending on the scenario. Each of the scenarios correspond to a different use case and a parallelization method, but are not exclusive. Both can coexist at the same time in the same network.

Presenting the designs and behaviour for these two scenarios is the main task of this document. The difference between them relies on the distinct communication architectures proposed, and hence the node kinds that each of them presents.

2.2.1 Collaborative learning scenario

Different AML-IP-Engine nodes can work on the same problem independently using different training datasets, and share the knowledge learned with other nodes. This creates a more complex and accurate model. The output data being shared is independent of the dataset used to train the algorithm, so the training datasets can remain private while the resulting knowledge may be openly published and shared with the rest of the network. The AML-IP provides a collaborative and Internet wide network so remote and independent nodes have the possibility to work on the same problem and share their progress.

2.2.2 Workload distribution scenario

AML-IP-Engine nodes are in charge of training an AML model by processing chunks of data. This training process is expensive and requires high computational power, but it is also completely parallelizable. The AML-IP provides a network architecture that is able to distribute this workload amongst the nodes available in the network in a balanced and automatic way regardless of their location and ownership, thus avoiding the need for centralized load balancing.

2.3 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 work 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.

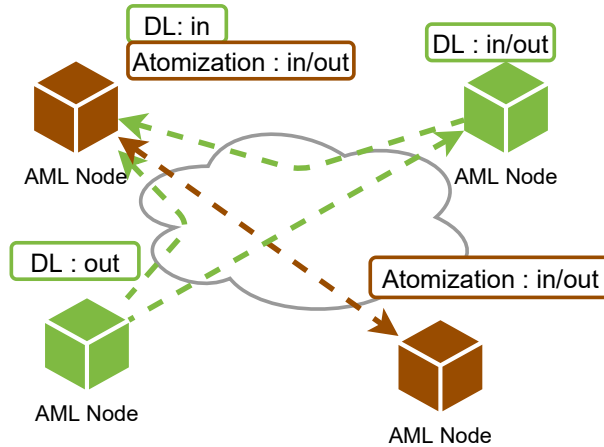


Figure 1: AML Nodes communicating with each other.

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 AML-IP Algebraic Description Language Node (AML-IP-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.

2.4 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.

2.5 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.

2.5.1 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 eProsima 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 (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 2.4. 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.

2.5.2 eProsima Fast DDS

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

eProsima Fast DDS is a C++ implementation of the DDS (Data Distribution Service) Specification, a protocol defined by the Object Management Group (OMG). The eProsima 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. eProsima 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.*

2.5.3 Fast DDS Python Bindings

eProsima has recently developed a library that enables the use of Fast DDS from within a Python environment by binding the C++ library. This is a very useful feature for the ALMA project, as most of the code produced for AML is written in Python, and Python will most likely be the chosen language for the main framework with which the end users will interact.

2.5.4 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 AML-IP Architecture

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

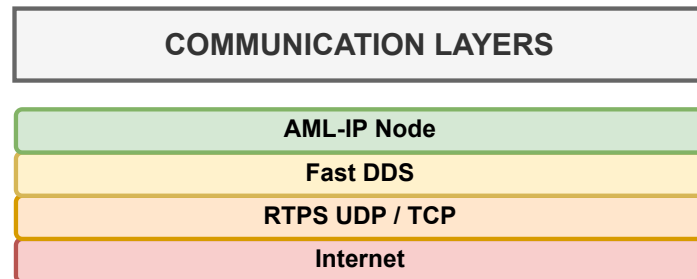
scalability and emerging requirements.

3.1 AML-IP architecture description

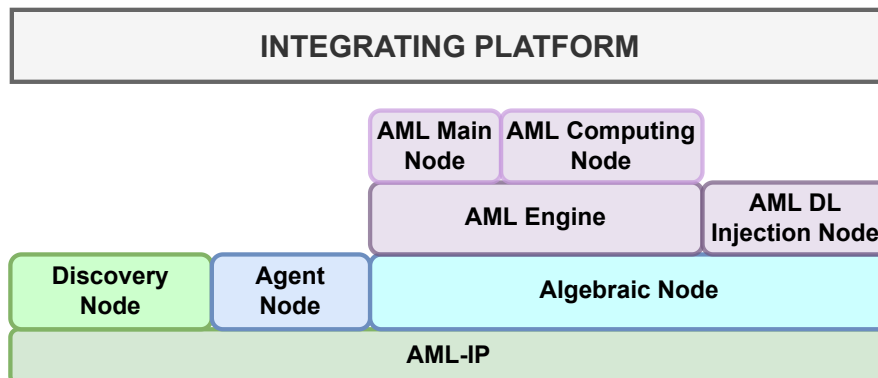
The approach this design has followed is focused on the division of the different AML-IP components that constitute the network. In order to divide the computational effort, to distribute the communication process and to create an scalable and widely usable architecture, the components of the AML-IP 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):** These components will be disseminated along the network and made 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 general architecture of the AML-IP is displayed in Figure 2. It shows how the AML-IP would integrate all transport and network elements (2a), as well as all types of AML-IP nodes and the hierarchy among them (2b).



(a) AML-IP communication layer model.



(b) AML-IP nodes layer model.

Figure 2: AML-IP Architecture

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 Nodes (AML-IP-Ds) 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-Agents, 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.

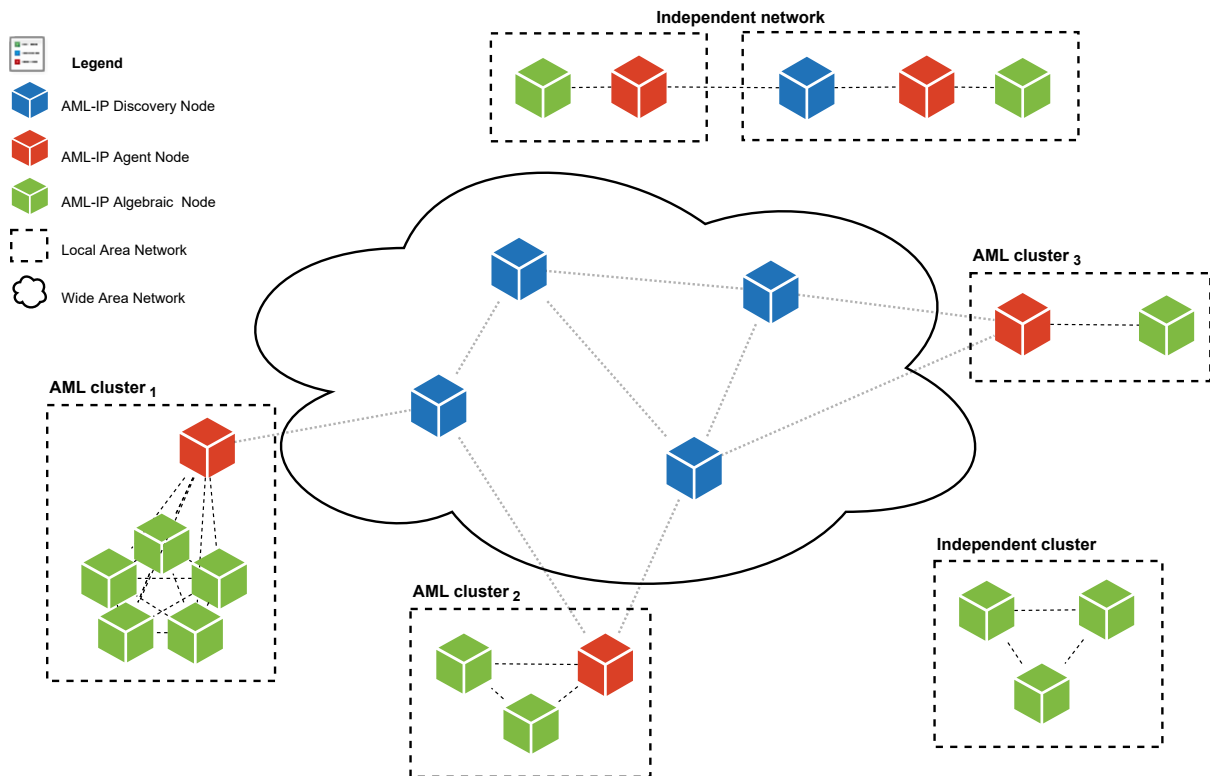


Figure 3: AML-IP example in LAN and WAN networks.

3.2 Use Cases

The AML-IP is a communication framework focused on the easy 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.

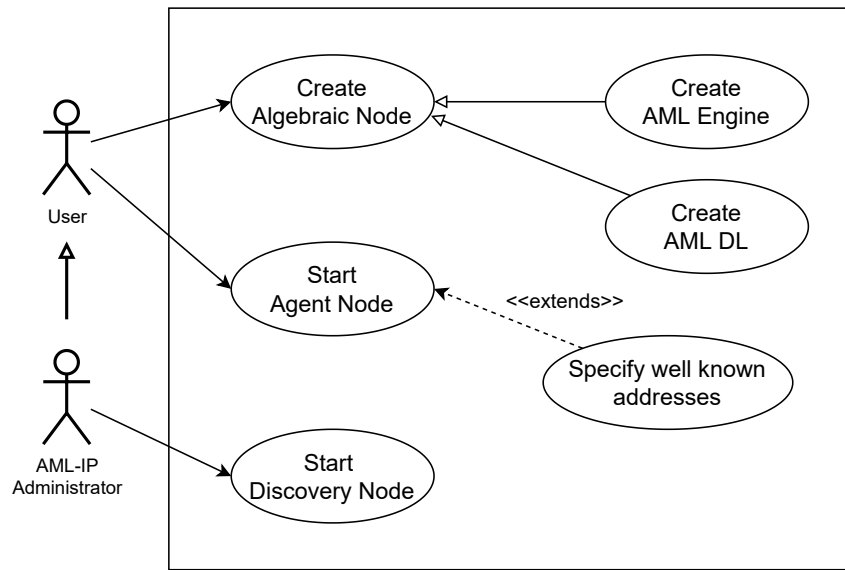


Figure 4: Use case diagram.

3.3 AML-IP functional operation

This section explains and exemplifies 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.

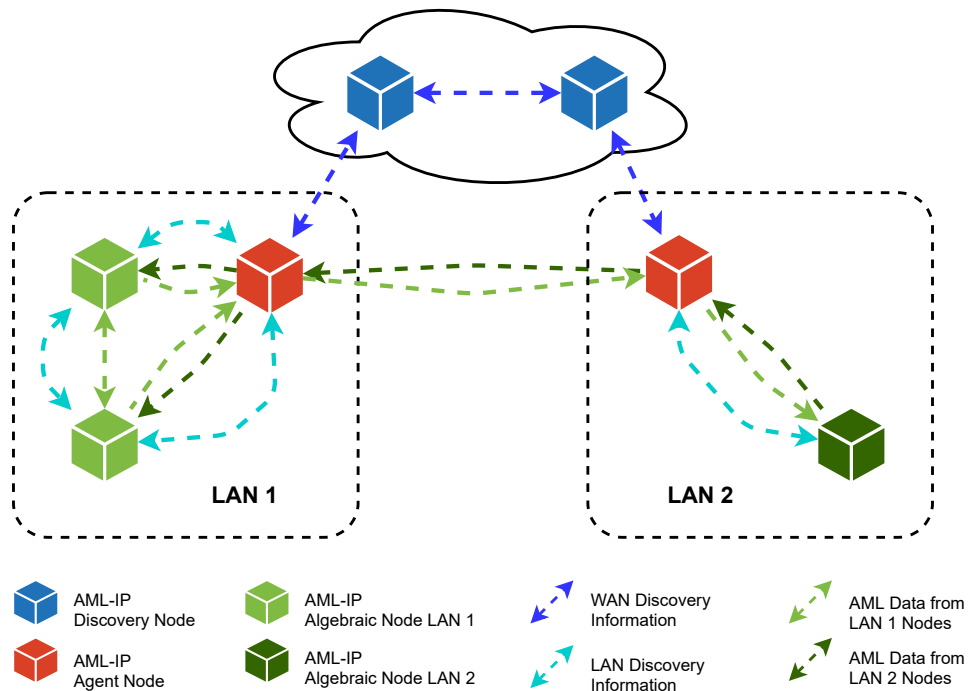


Figure 5: AML-IP minimum example to describe LAN and WAN communication.

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.

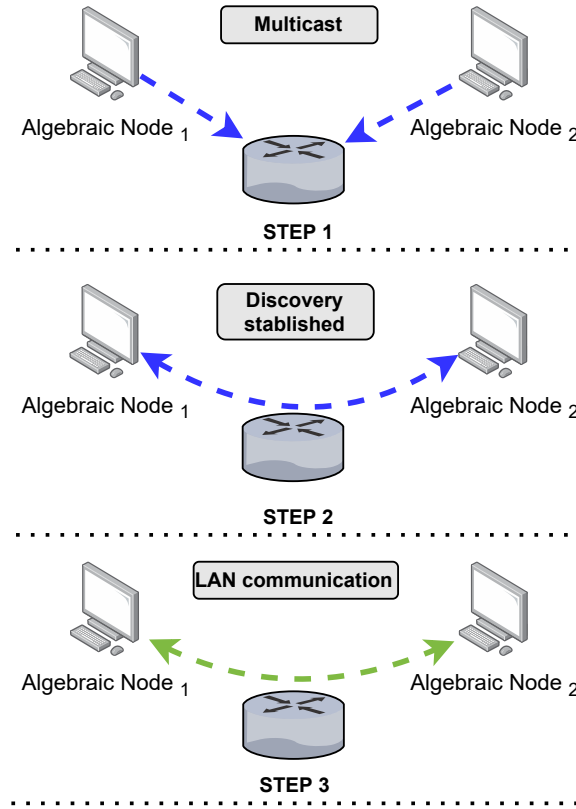


Figure 6: LAN communication schema.

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.

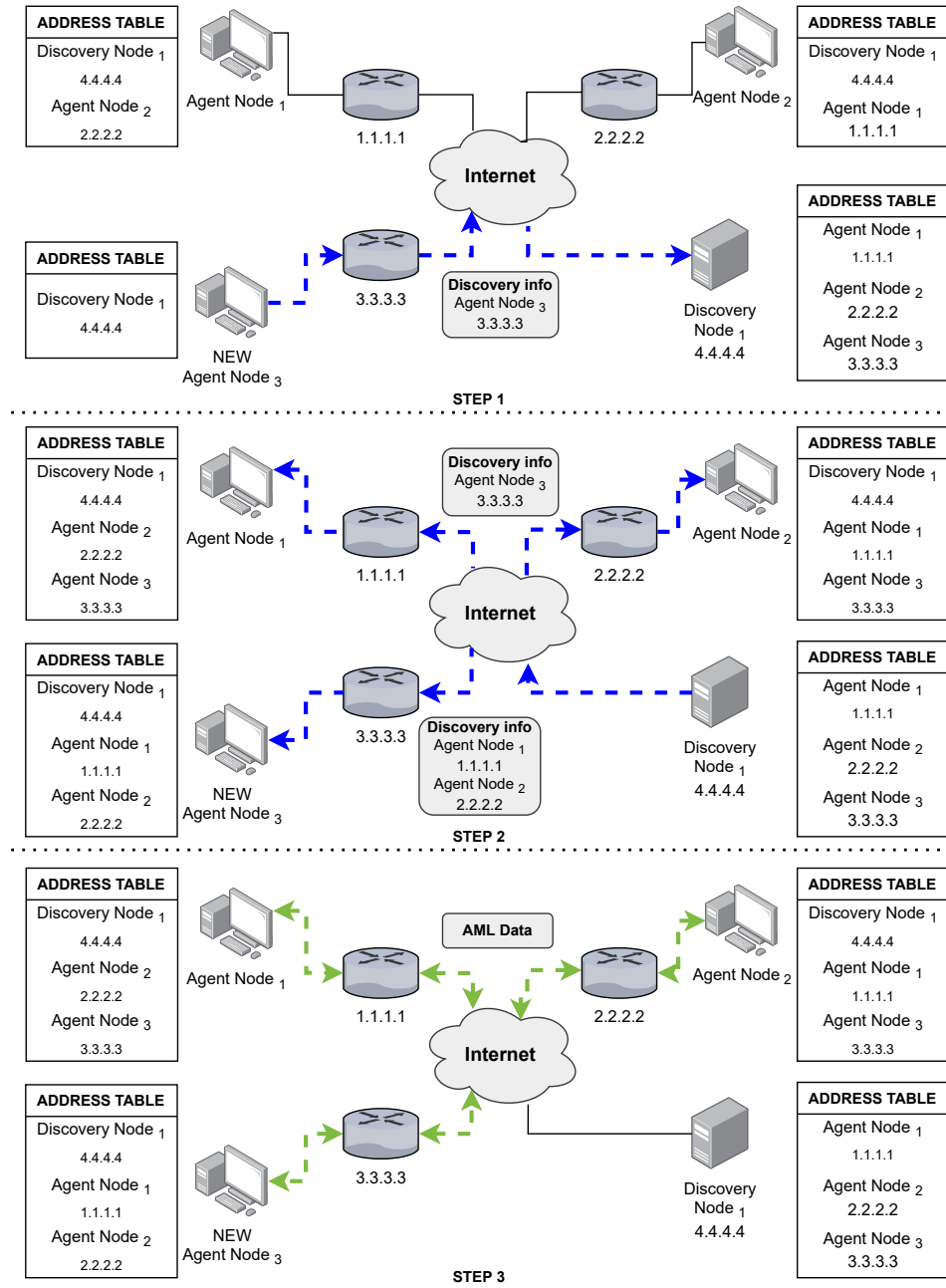


Figure 7: WAN communication schema.

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.

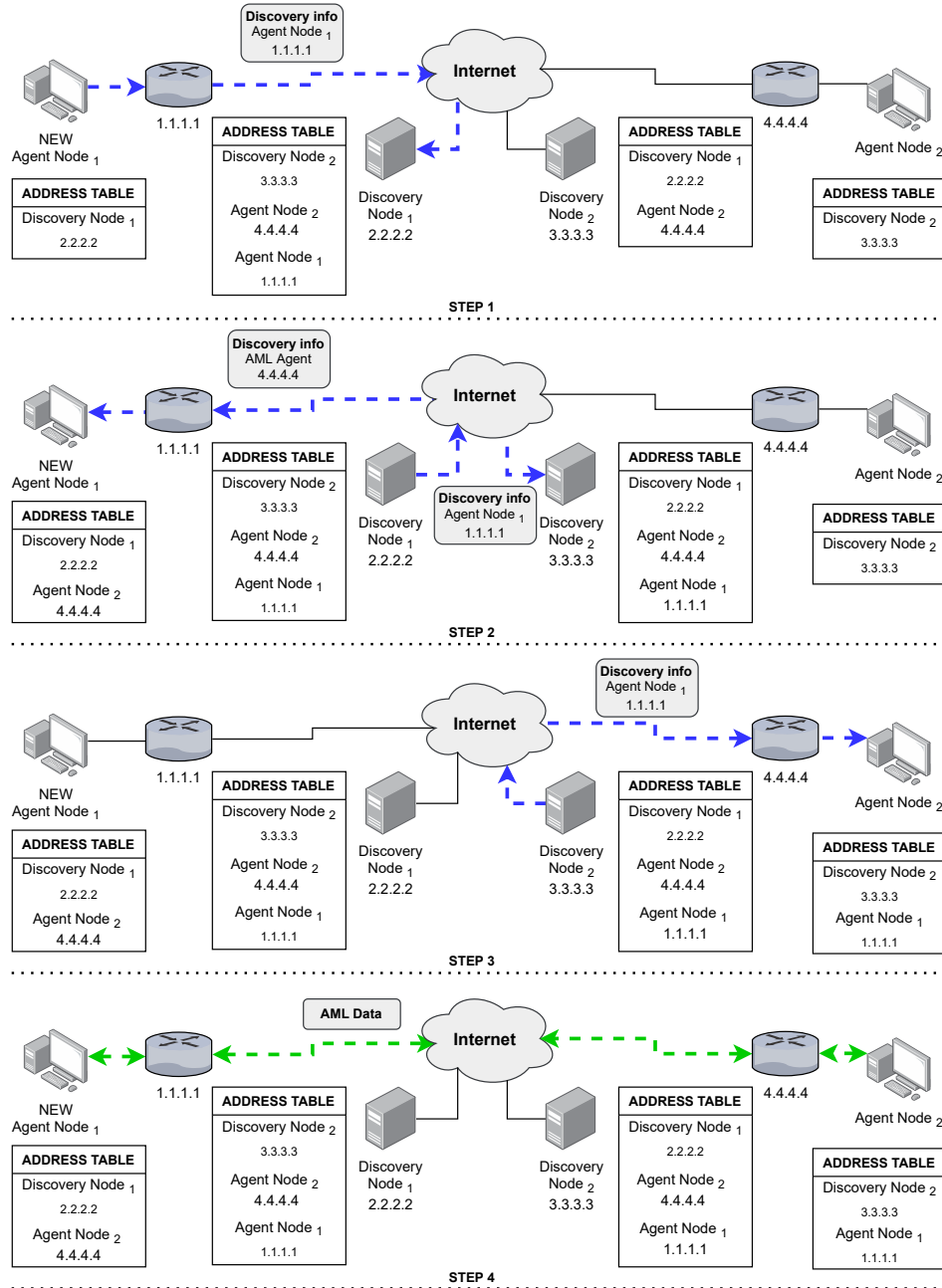


Figure 8: WAN communication schema with a network of AML-IP-D.

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.

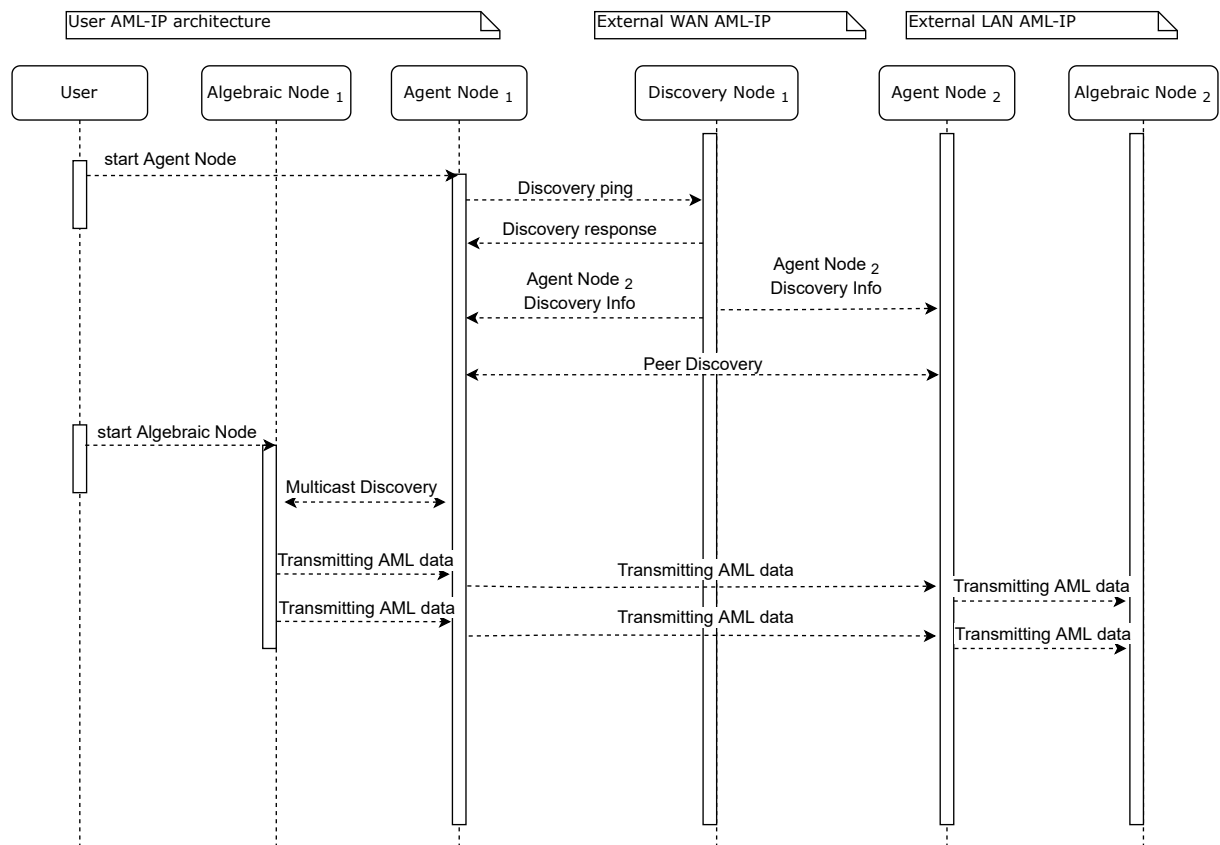


Figure 9: AML-IP sequence diagram in LAN and WAN networks.

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 Algebraic 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.

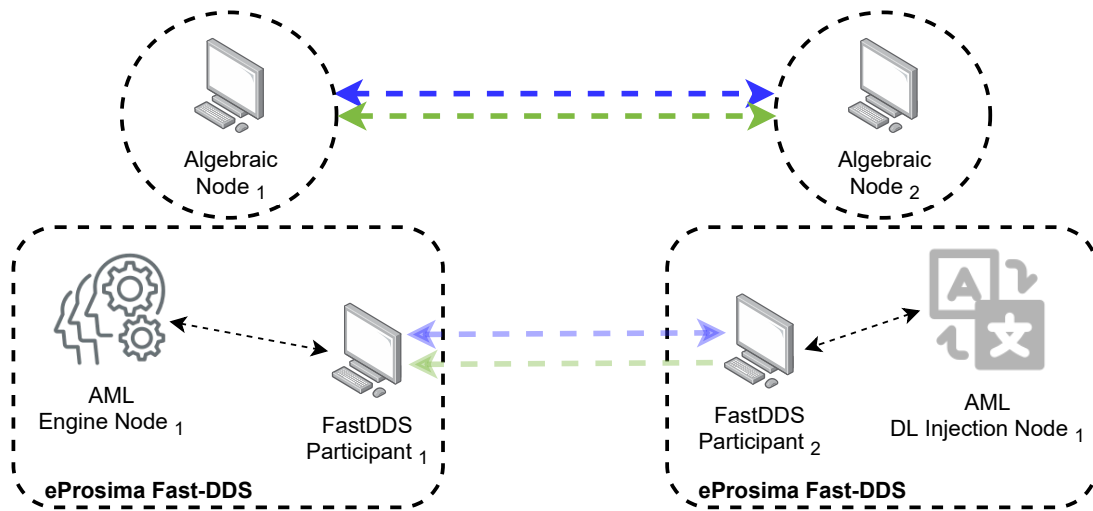


Figure 10: AML-IP Algebraic Node (AML-IP-Alg) design.

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 2.5.2 describes *Fast DDS* in detail.

The design, implementation and operation of the AML-IP-Alg is a work in progress, and will certainly be subject to change as the project advances. Two subtypes of AML-IP-Alg nodes, the AML-IP-Main and AML-IP-Computing are introduced in section 5, along with the description of the role each play in the workload distribution scenario. Some other types of algebraic nodes, as the AML-IP-Engine, the AML-IP-DL and new ones that might emerge in future versions of the AML-IP, will be defined in subsequent revisions of this document.

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 routing of data, *b)* the discovery of AML-IP components in WAN, and *c)* the inclusion of mechanisms to make the AML-IP components accessible from outside the LAN using UPnP.

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

- It greatly facilitates 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 leverages **eProxima Fast DDS Discovery Server**, which automatically creates 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.

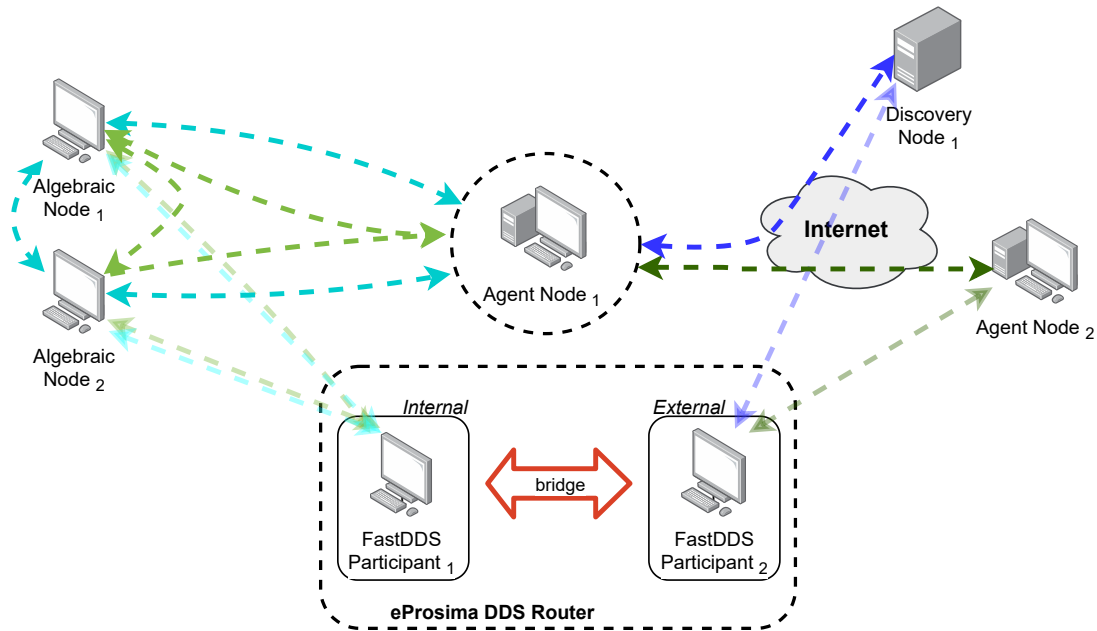


Figure 11: AML-IP Agent Node design.

a) DDS Router

The AML-IP-Agent centralizes the communication in the same LAN to be sent and received in WAN. That is, every message that needs to be sent outside the LAN or be received from outside must pass through it. This task is performed by the eProsima DDS Router.

The DDS Router is a new application implemented by eProsima specifically designed for the AML-IP to communicate DDS networks in WAN. This application launches several DDS Participants (generally 2) that communicate with different DDS networks given their difference in configurations. The DDS Router creates a bridge between pairs of participants and transmits every data received by one of the participants to the rest of them, allowing isolated networks to communicate with each other. A simple diagram showcasing the role of DDS Router in the AML-IP can be found in Figure 12.

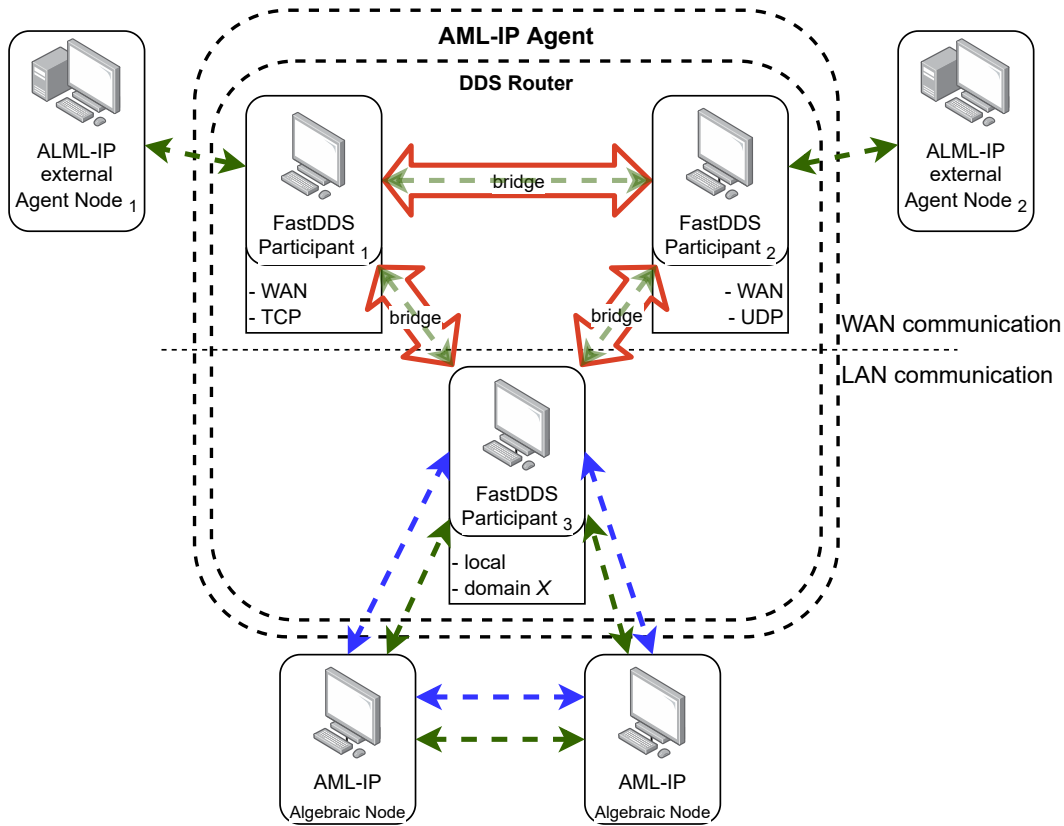


Figure 12: AML-IP Agent Node (AML-IP-Agent) powered by DDS Router.

Hence, the DDS Router allows the AML-IP-Agent to serve as a communication bridge between an internal (LAN) network and an external one. Every independent LAN would then count with an AML-IP-Agent that transmits internal data to external AML-IP-Agents, which then transmit this external data to their internal nodes. This way, a communication flow between every node is achieved regardless of whether they are in the same or different LANs.

The DDS Router is an Open Source software developed by eProsimia and hosted in a Github repository ¹ under the Open Source Apache 2.0 license.

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

¹<https://github.com/eProsimia/DDS-Router>

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 13 shows an example of how an AML-IP-Agent uses UPnP to be accessible to other AML-IP-Agent in 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 stablished.

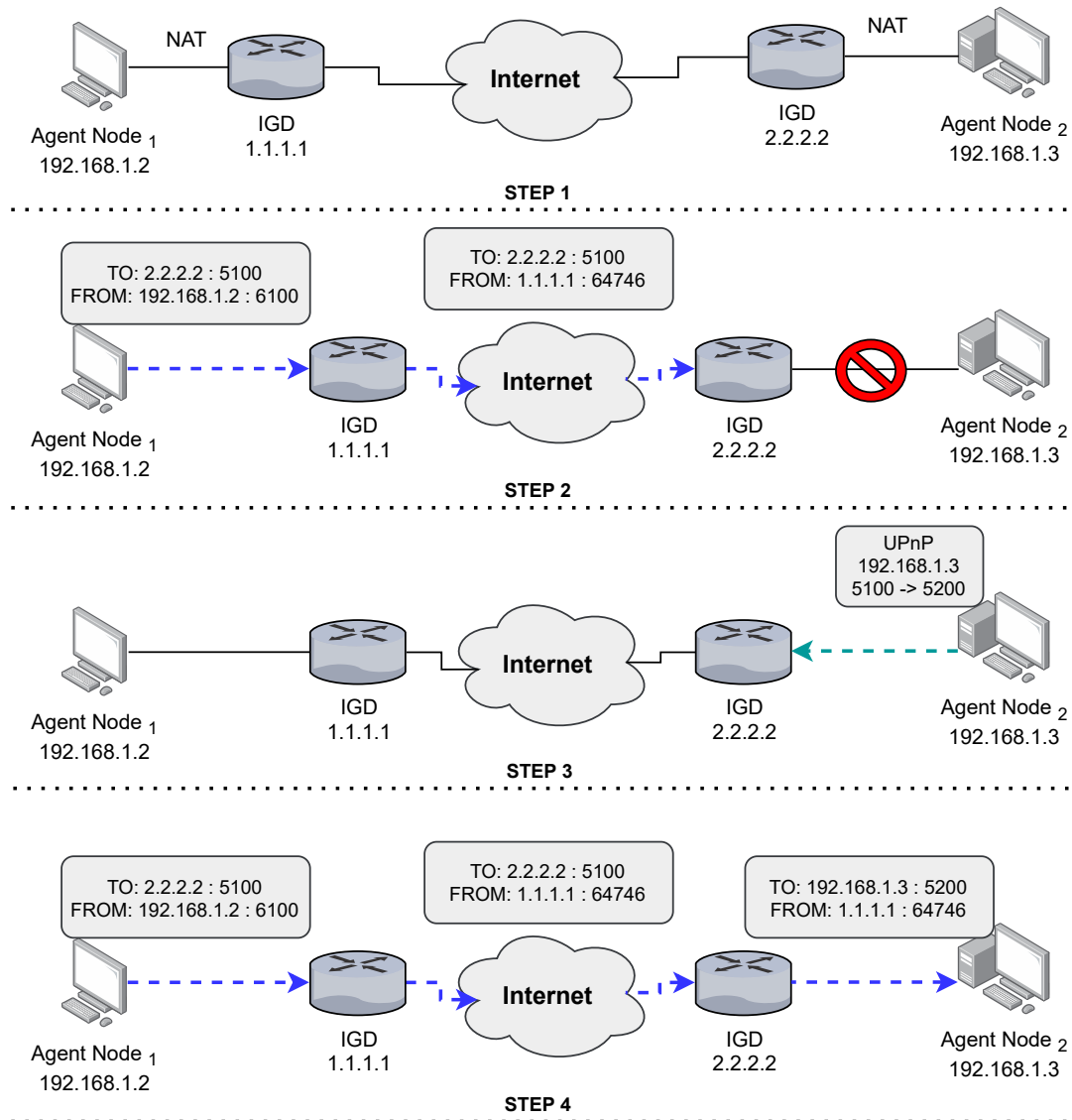


Figure 13: UPnP execution example.

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 2.5.2, 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 14. The AML-IP-D implements a *Fast DDS* Discovery Server, sharing the discovery information of the DDS entities that connect with it.

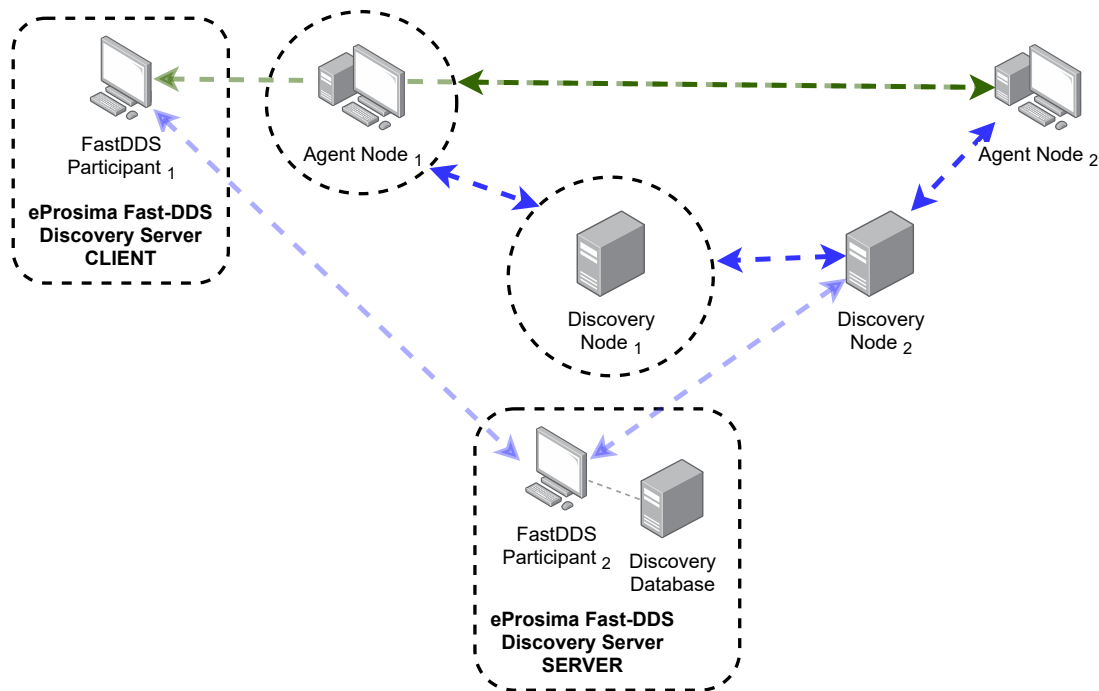


Figure 14: AML-IP Discovery Node design.

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 in a constant state of evolution, many of its components 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 AML-IP architecture scenario 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**² under the Open Source Apache 2.0 license ³.

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:

²https://github.com/eProsima/aml-ip-prototype/tree/ALMA_AMLIP_D6.1

³https://github.com/eProsima/aml-ip-prototype/blob/ALMA_AMLIP_D6.1/LICENSE

- **AML-IP-DL:** These nodes will represent every node in the network that publish under a *Description Language* topic. The output of these nodes will be *AML-IP-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.
- **AML-IP-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 AML-IP-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.
- **AML-IP-D:** These nodes will execute the discovery process that joins 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
AML-IP-DL	DL		<code>_aml_ip_topic.dloutput</code>	<i>AML_IP_DLOutput</i>
AML-IP-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>
AML-IP-D				

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 2.3, 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 AML-IP-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 AML-IP-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.

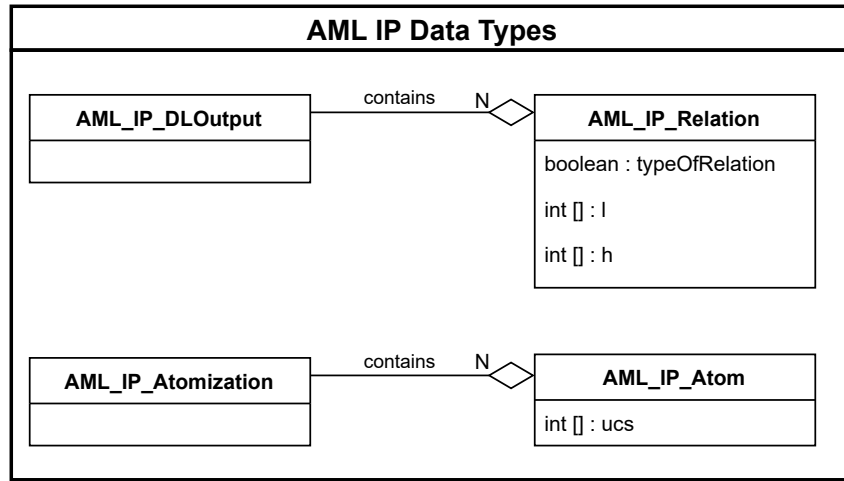


Figure 15: Data Models.

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*⁴. 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 15 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
  
```

⁴<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 AML-IP-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`⁵. In this same repository there is also an example of how the same AML-IP-Engine node subscribes to and receives data from a specific topic^{6 7}.

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 2.5.2, *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 Docker⁸ containers.

⁵https://github.com/eProsima/aml-ip-prototype/tree/ALMA_AMLIP_D6.1

⁶AML-IP-Engine publication example:
<https://github.com/eProsima/aml-ip-prototype/blob/c6d6de24de7b3201082cb53efba49e01e733c09f/src/cpp/Engine/EngineParticipant.cpp#L240>

⁷AML-IP-Engine subscription example:
<https://github.com/eProsima/aml-ip-prototype/blob/c6d6de24de7b3201082cb53efba49e01e733c09f/src/cpp/Engine/EngineParticipant.cpp#L342>

⁸<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 1⁹:** Test static discovery using TCP connection between AML-IP-DL node and AML-IP-Engine node.
- **PoC 2:** Test the connection of AML-IP-DL and AML-IP-Engine nodes, each in different LANs, connected to a AML-IP-D 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 AML-IP-DL and an AML-IP-Engine nodes, and another machine running an AML-IP-Engine and a AML-IP-D 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 16 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 AML-IP-D, 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 AML-IP-D that knows beforehand the address of the on already in the net. The AML-IP-Engine and the AML-IP-DL will connect to their respective AML-IP-D 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.

⁹This 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.

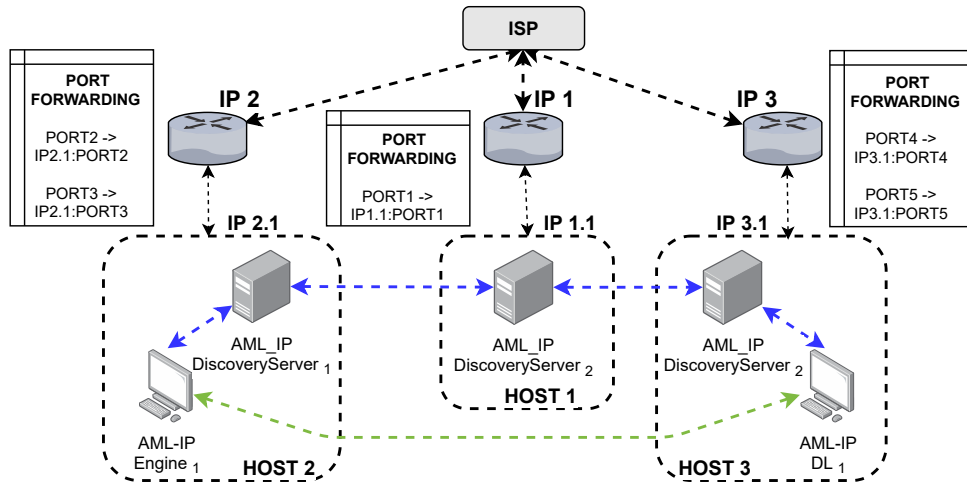


Figure 16: Experiment architecture.

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 repository¹⁰. The project has been developed for Ubuntu 20.04.2.0 LTS (Focal Fossa)¹¹ 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`¹² 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`¹³ installed, which is not supported by the `apt`¹⁴ 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/ALMA_AMLIP_D6.1/amlip.repos
$ vcs import src < amlip.repos
$ colcon build
```

¹⁰https://github.com/eProsima/aml-ip-prototype/tree/ALMA_AMLIP_D6.1

¹¹<https://releases.ubuntu.com/20.04/>

¹²<https://colcon.readthedocs.io/en/released/>

¹³https://www.boost.org/users/history/version_1_71_0.html

¹⁴<https://wiki.debian.org/Apt>

4.3.3 Experiment setup

For the description of the experiment setup, the terminology of Figure 16 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 30.

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

• HOST 1

This host executes a single AML-IP-D 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 16. 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 AML-IP-D.

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 AML-IP-D node in **HOST 1**.

where

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

• HOST 2

This host executes an AML-IP-D that communicates with the AML-IP-D deployed in **HOST 1** and listens in **localhost** in order to discover every node working on **HOST 2**. It will also execute an AML-IP-Engine that will listen to every AML-IP-DL node publishing DL data under the **AML_IP_DLOutput** topic and to every AML-IP-Engine publishing Atomization data under the **AML_IP_Atomization**. In turn, this AML-IP-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 15.

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 16. 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 AML-IP-D 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 AML-IP-D node in HOST 2.

where

- **id** sets this AML-IP-D 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 AML-IP-D with which this AML-IP-D is going to connect; and
- **time** set to 0 implies that the AML-IP-D will be running until the **Enter** key is pressed.

Finally, the HOST 2 administrator runs the AML-IP-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 AML-IP-Engine node in HOST 2.

where

- **listening-addresses** sets the addresses to which the AML-IP-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 AML-IP-D 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 AML-IP-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 AML-IP-DL instead of an AML-IP-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 16. 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 AML-IP-D 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 AML-IP-D node in **HOST 3**.

Finally, the **HOST 3** administrator run the AML-IP-DL with the following command, similar to the one used for running the AML-IP-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 AML-IP-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 31 represents the execution of the first AML-IP-D in **HOST 1** after executing 1. In this screenshot could be seen the different AML-IP-Ds that connect to it and the discovery information of the nodes. This information is reported by all the AML-IP-DL and distributed to other AML-IP-DLs. Finally the two remote AML-IP-Ds 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 32 shows how the AML-IP-D in **HOST2** connects and discovers the central AML-IP-D deployed on **HOST1** after running the command 2. Once the local machine is already connected with the network of AML-IP-Ds, a new AML-IP-Engine is instantiated by running the command 3 in **HOST 2**.

Finally, in **HOST 3** the same deployment configuration is performed by executing an AML-IP-D with command 4 and a AML-IP-DL with command 5. It could be seen the execution of this AML-IP-D in

HOST 3 in Figure 34.

Once everything is deployed, the two nodes will discover each other by the network of AML-IP-Ds and will start sending and receiving data. This could be seen in Screenshot 35, which shows the data being sent by the AML-IP-DL. In the Screenshot 33 it can be seen how the AML-IP-Engine publish its own kind of data, but since there are no subscribed nodes, none receives it. Furthermore, it could be seen how this AML-IP-Engine receives data from a remote AML-IP-DL, which is in this case in HOST 3.

The following details how to understand the output of an AML-IP-Engine and AML-IP-D execution in the prototype (Figures 35 and 33). 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 AML-IP-Engine node and the AML-IP-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 AML-IP-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 AML-IP-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 AML-IP for workload distribution scenario Design

This section explains the different types of nodes that participate in the AML-IP for workload distribution scenarios, and the communication between them.

5.1 Workload distribution scenario

The AML-IP is a platform designed to cover a wide range of actions in an AML network; it allows for the collaboration between different networks, keeping the privacy and independence of each node; it makes possible to parallelize certain tasks between several local or remote nodes; it allows to merge several trained models in order to obtain a better one; and it also contemplates those use cases where multiple nodes may work jointly on the same task, by facilitating the communication and synchronization between nodes.

The so called *workload distribution scenario* refers to those scenarios in AML where one computational complex task could be divided in subtasks, and these subtasks could be processed in parallel by different computational points, to finally obtain a joint result.

The scenarios are not mutually incompatible, they just differ in the kinds of nodes that participate in each of them. A workload distribution scenario could be present locally, remotely, or even both at the same time. By leveraging the versatility of the AML-IP-Agent, an AML-IP-Engine could be working distributively and at the same time collaborating with a network of AML-IP-Engines by sharing its training result. The most important nodes in this scenario are the AML-IP-Main and the AML-IP-Computing, which are explained ahead in this section.

Training parallelization The workload distribution scenario directly applies to the training process performed in the AML-IP-Engine, the one in charge of training a model by adding new atomizations resulting from the processing of batches of data. This is a very expensive process that requires a high amount of memory and computational power. But it is also a parallelizable process, where some batches of data could be trained in parallel, and then the results could be merged to obtain a better model.

5.2 AML-IP for workload distribution scenarios

This section explains in detail the design of certain nodes of the AML-IP to achieve the efficient deployment of a training network exploiting the strengths of AML.

5.2.1 Architecture Description

The main idea of this part of the AML-IP is to parallelize the computational effort in a balanced way between an arbitrary number of computing points. The work that is presented along this section refers to the process of creating new atomizations, generated by an AML-IP-Engine. It could be said that the scenario here exposed comprehends the detailed process of an AML-IP-Engine distributing its own workload among a number of computing points.

The nodes that take part in this sort of scenarios are the following:

- **AML-IP-Main:** This kind of node functions as a central point for a *training* process. It creates *Jobs*, that could be seen as batches of data to be processed by the training algorithm. These jobs are distributed to a number of computing nodes (as any batch could be trained in parallel thanks to the AML), and their corresponding results collected and merged back in the main node.
- **AML-IP-Computing:** This kind of node is in essence a computational point where a workload is released, so it can process a specific task on the data it has been given. This task corresponds to training over a batch of AML data and sending the pertinent result to the AML-IP-Main.

Figure 17 represents the previously described scenario. In this example, two AML-IP-Main work in the same problem, but with different datasets. Each of them has two different batches of training data to process, but since an initial preprocessing phase is needed, the two batches are not required to be executed at the exact same time. With this schema it can be accomplished that, using three AML-IP-Computing,

the workload of the two AML-IP-Main is distributed among them in an efficient way, so a task being resolved by a machine and other executions do not overlap. This is a specific use case, but any scenario with an arbitrary number of AML-IP-Main and AML-IP-Computing nodes is possible, and the workload would be *intelligently* divided between available and free nodes so the parallelization of complex tasks (as, in this case, the training phase) is maximized.

In this example, the *collaborative learning scenario* use case where the two AML-IP-Main nodes share their training results (atomizations) to obtain a better one is also represented. This is only to highlight that both scenarios are not mutually exclusive.

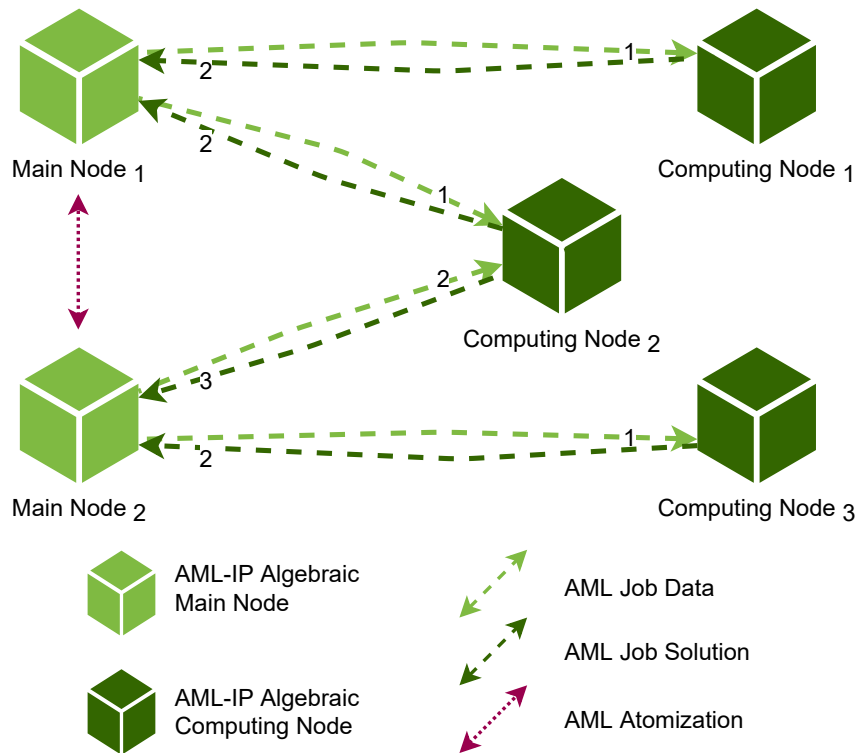


Figure 17: AML-IP workload distribution scenario.

It is important to make clear that a node is an abstract software entity that exhibits a certain behaviour, but it is not directly related with a physical machine. Thus, there is no need to have access to cluster of machines in order to run this software, an AML-IP-Main and several AML-IP-Computing could be running in the same machine (computer) at the same time. This may not be the most optimal scenario, but it still benefits from modern processors' multi-threading tasking or specific hardware advantages.

5.2.2 Communication

Every node is communicated via **DDS**, just like the rest of the AML-IP. This is a great advantage since the discovery is made automatically between the nodes, and the communication is completely dynamic and distributed. This allows to have a completely decentralized network where every node is independent of the network status. (e.g. a AML-IP-Main node will behave the same way if there are 0 or 10 AML-IP-Computing nodes in its network. Obviously, in the first case it would not be able to train a model, and in the second it would train much faster than with just a AML-IP-Computing node).

In the case where different nodes are not in the same LAN, we would mix this scenario with the *collaborative scenario*, where a node could communicate via WAN with external nodes hosted anywhere by using the AML-IP-Agent.

5.2.3 Synchronization

It is clear that a way of synchronizing the different nodes is necessary in order to achieve a correct parallel training, being the nodes independent between each other. This is accomplished by using a new communication protocol on top of the DDS layer (application transport layer) that allows to manage a synchronous understanding between nodes using publication and subscription interfaces. This new protocol is called **MultiService Communication Protocol** and is explained in section 5.3.

5.2.4 Multiple scenarios

The AML-IP-Main and AML-IP-Computing nodes represent the main points in the context of a workload distribution scenario in an AML network. This architecture is very useful to parallelize the model training workload between several nodes. This scenario is represented in Figure 18, where a single AML-IP-Main could distribute the effort between 4 independent AML-IP-Computing. This way, the time it takes to train a model is inversely proportional to the number of AML-IP-Computing in the network.

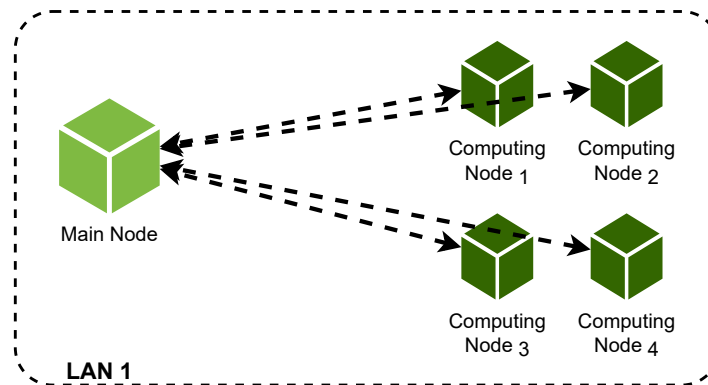


Figure 18: AML-IP local parallelization scenario.

However, this scenario is not the only possible one. It would also be possible to use these parallelization capabilities in a WAN scenario, where one or many AML-IP-Main could benefit from having a shared external cluster of AML-IP-Computing. This could be achieved by deploying a AML-IP-Agent in each of the different LANs. This scenario is represented in Figure 19.

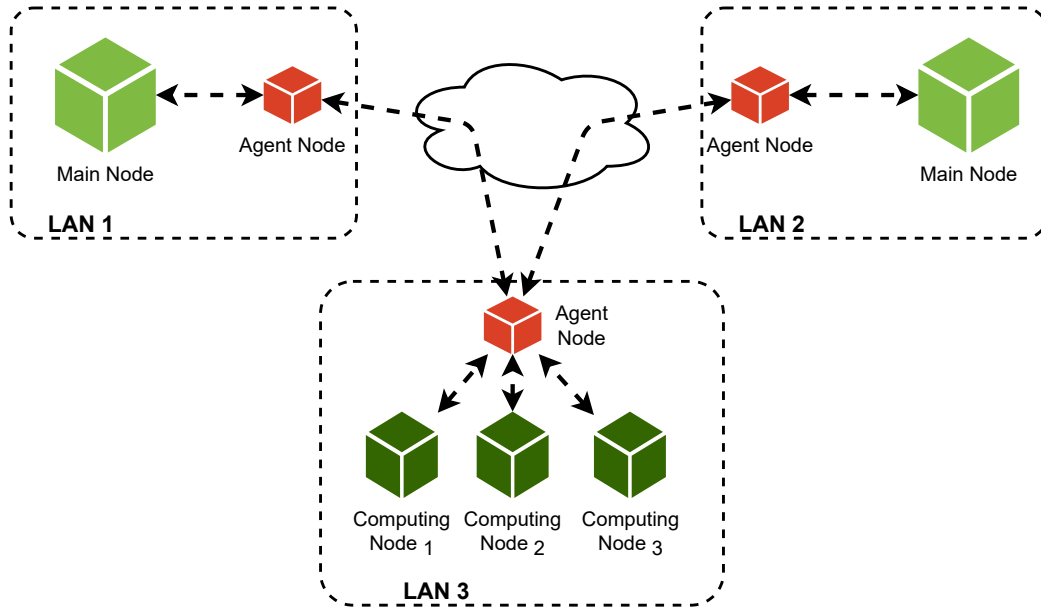


Figure 19: AML-IP WAN parallelization scenario.

And finally, to exemplify that it is possible to mix the AML-IP workload distribution scenario with any other, in Figure 20 it is represented a scenario where one AML-IP-Main trains a model using a local AML-IP-Computing and shares the resulting model with an external AML-IP-Main unable to train its own model (say due to the lack of computational resources).

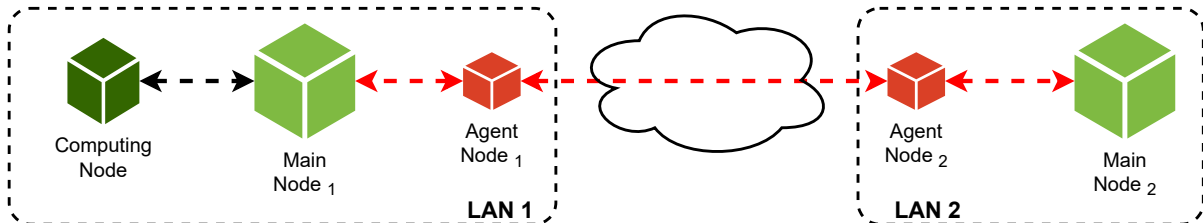


Figure 20: AML-IP workload distribution and collaborative learning scenario.

5.3 MultiService Communication Protocol

The MultiService Communication Protocol (MSCP) is a new communication interface or protocol that exploits the advantages of DDS to create a specific protocol suitable for AML.

This new protocol has been designed by eProxima in order to fulfill the data transmission requirements for the different AML-IP-Alg, and it is not a final design but a work in progress that may be updated in the future for a more robust and efficient one.

5.3.1 Description

MSCP is a new protocol based on DDS that responds to communication necessities specific to the ALMA project. This protocol targets the communication between several clients and servers in a way that only

one server processes a request from any client, thus avoiding the use of a load balancer. This allows to parallelize certain computationally expensive tasks of a client in different servers.

Diagram 21 represents the general schema of the communication network that MultiService establishes. There are an arbitrary number of clients and an arbitrary number of servers, and each of them are independent of each other. The clients send their requirements, and one and only one server in the network will process that requirement and return a solution for it.

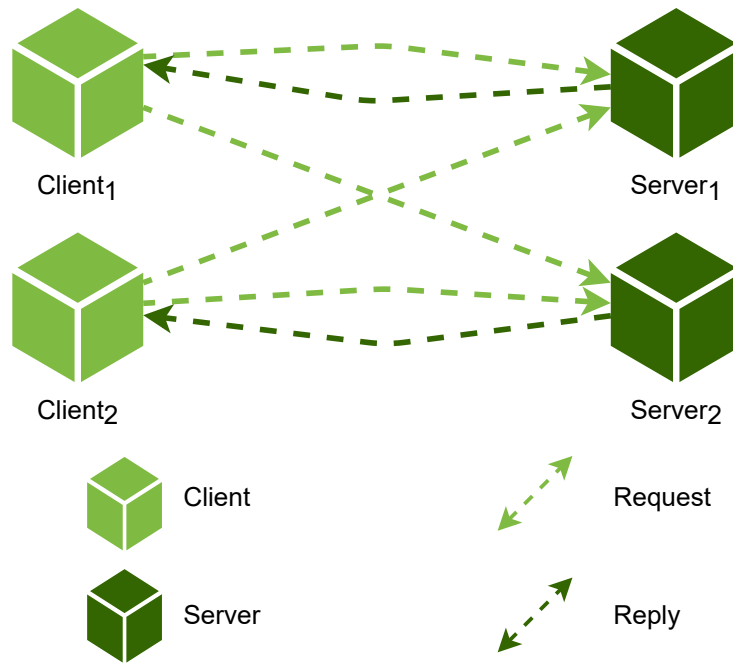


Figure 21: MSCP Schema.

5.3.2 Requirements

These are the requirements that the protocol must fulfill:

- **Remote procedure:** There must be nodes (*clients*) that create **tasks** (usually complex computational problems) and distribute them to other nodes (*servers*) that process them and return a result.
- **Node independence:** The behaviour of each node must be independent of the network where it is running. (e.g. a client must behave the same way if there are none, one or three servers in the network).
- **Dynamic network:** New nodes may enter and leave the network at any time and with any network address.
- **Efficient data distribution:** It must keep the amount of data transmitted over the network to the minimum. It must eschew duplicated or useless data transmissions. (e.g. Clients must avoid sending data to servers that will not process their requests).
- **Intelligent workload distribution:** The network must efficiently distribute the tasks between the servers available at that moment.

- **Fully distributed:** Avoid the use of central points so there is no bottle necks or single-point-of-failure nodes in the network.

5.3.3 Design

The idea behind the MSCP is to create an *intelligent* load balanced network using only DDS communication between different nodes. For this purpose, a series of topics, QoS and data types have been created in order to accomplish such goal.

In sequence diagram 22, it is shown the different messages that are exchanged between client and servers in order to achieve the communication of the data task and the answer with its solution.

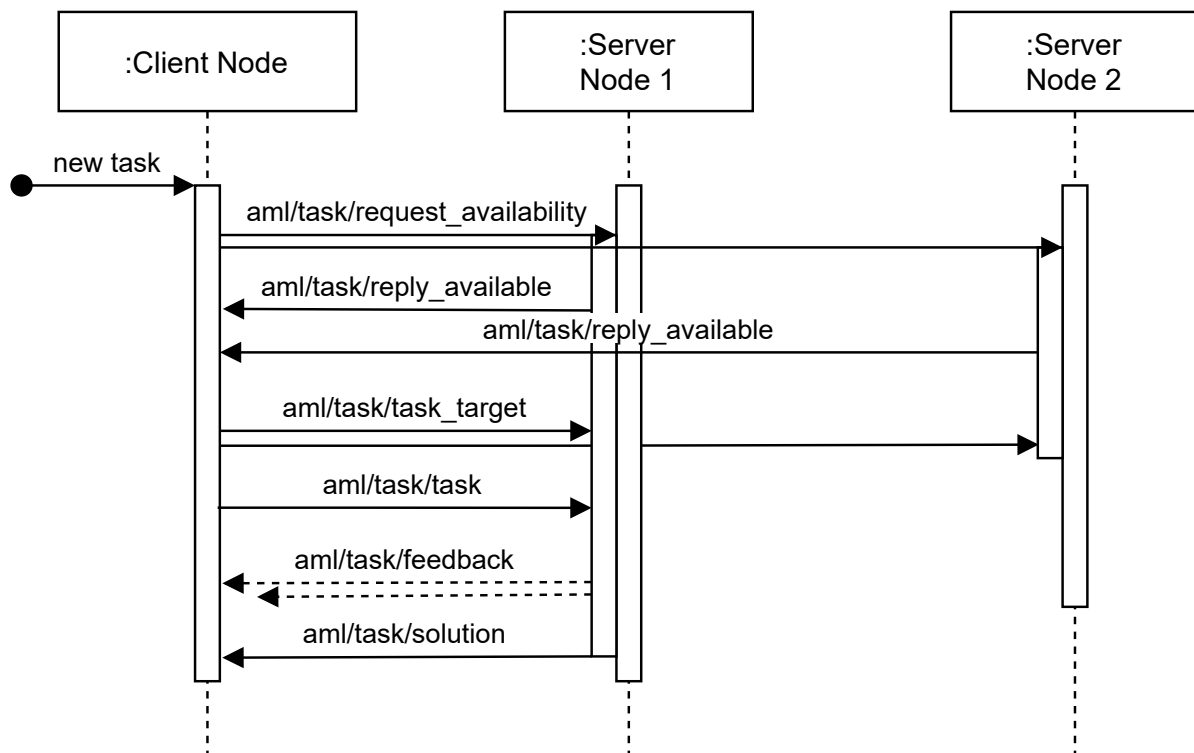


Figure 22: MSCP sequence diagram.

Figure 23 shows the different steps in the protocol communication between a client and a server. Following are the steps of the communication process:

- **1** First a client publishes a request asking for any available server.
- **2** Whenever a server is available and receives such request, it publishes a message reporting its availability for the task under consideration. After a server has replied, it remains blocked waiting for the confirmation from the client.
- **3** Once the client has received an availability reply from one of the servers, it marks it as the *target server*. That is, the server that will be performing the client's task. This client publishes under a specific topic that this task is to be completed by the *target server*, communicating to this server that it is the one in charge of doing the work, and to the others that they are no longer required for the task at hand. Just after sending this message, the client sends the data relative to the task

only to the *target server* (this could be achieved in DDS by using different mechanisms such as *partitions* or *ContentFilteredTopic*).

- If a server is blocked in a specific task and it is not selected to be the *target server*, when the **task_target** message is received, it unlocks and becomes available for new requests. If a server is *target server* for a client, it awaits for the task data to arrive. Once it is received, the *target server* starts the procedure.
- **4** In case the *target server* fails or stops, the client will start the process again. For this reason, while the server is processing the task, there may be additional data being transmitted from the server to the client, announcing that the task is being processed (feedback). This would dissuade the client from thinking the request failed and start the process again.
- **5** Finally, when the server obtains the answer to the task it is sent back to the source client, which ends the communication process.

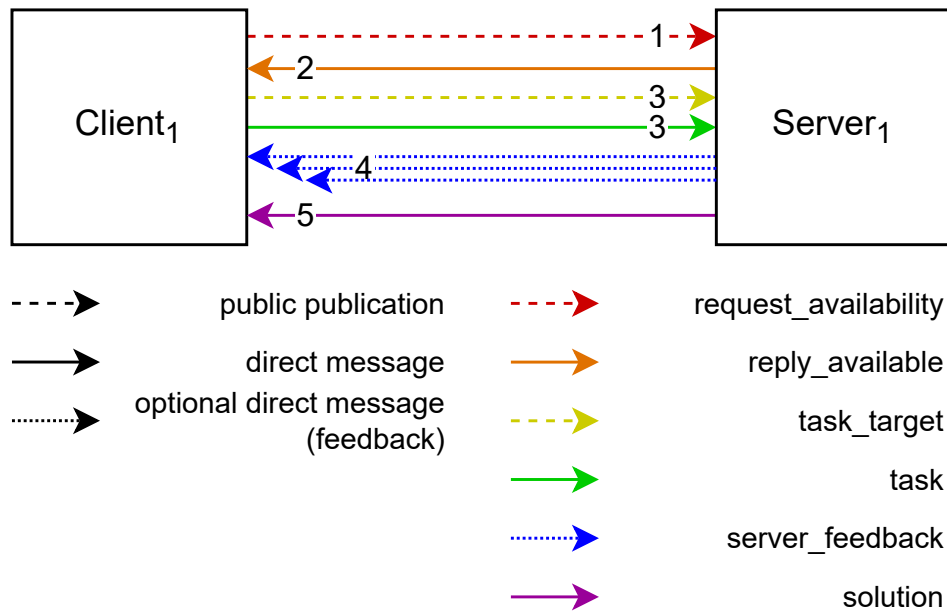


Figure 23: MSCP steps for single client-server communication.

In more complex cases involving several servers and/or several clients, the behaviour of each of them would be the same. As it is shown in image 24, the communication process is similar when a client tries to find a server to process its task. The client will ask for availability, and for the multiple answers it could receive, it would choose only one server and communicate that decision to all servers. The target server will process the task while the other(s) will, after leaving the blocking state, start processing other availability tasks that may have arrived in the meantime. As the figure evinces, the time spent by a server in a blocking state for a task that will not process (because other server will) is minimum, maximizing the time each server spends on execution procedures.

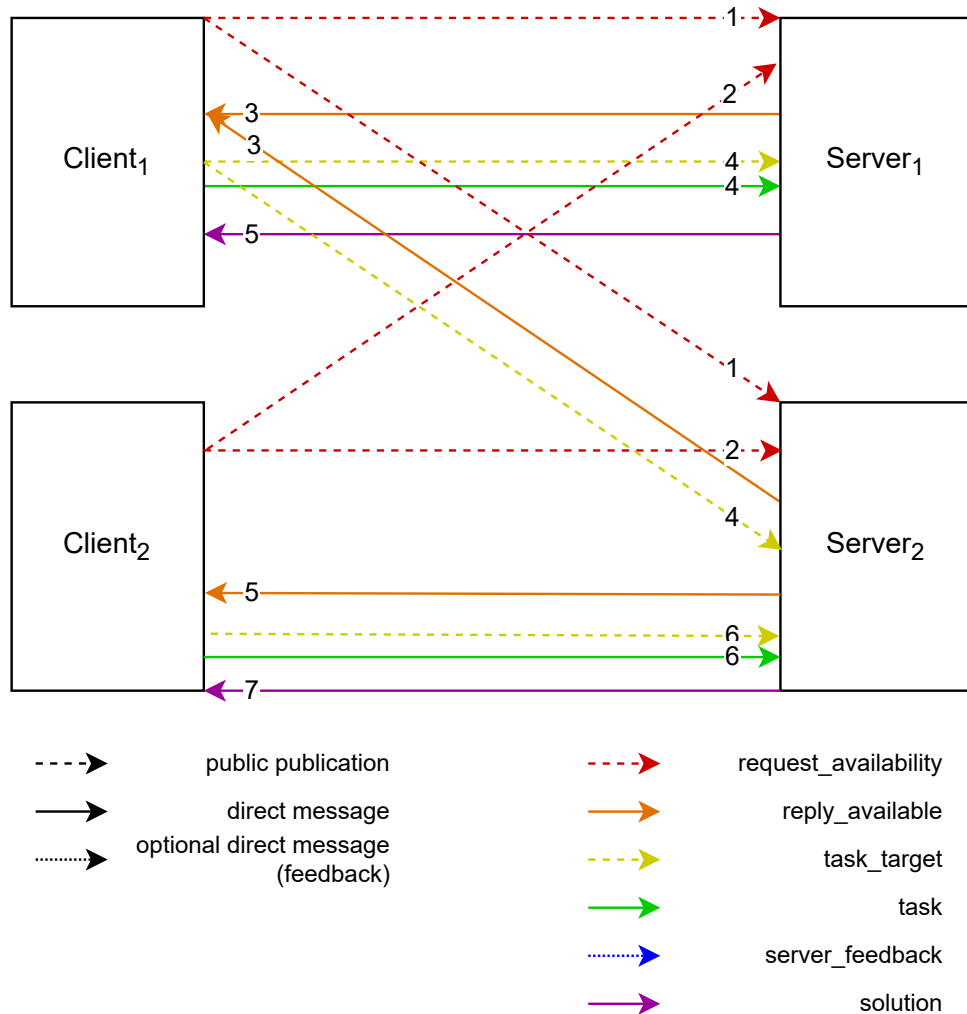


Figure 24: MSCP steps for multiple client-server communication. (In this figure, feedback messages have been removed for simplicity purposes).

5.3.4 Advantages and Limitations

The benefits that using MSCP offer are intrinsic to DDS, a fully distributed protocol, and the reliable synchronization between different nodes to decide which one processes each task. The main advantages that one could expect from this protocol (among others) are:

- **Fast response:** As can be seen in figure 23, there are only 3 meta-traffic messages needed in this communication protocol, avoiding extra handshaking or discovery messages of any kind.
- **Minimum blocking time:** The target server is decided upon the first reply listened by the Client. This means that, any other servers blocked for the same task will be rapidly informed that they are no longer required. Thus, the amount of time that a Server will be blocked for no useful purpose is minimum.
- **Automatic synchronization:** The synchronisation naturally arises from the protocol itself, and

so does the workload balance, without the need of external nodes or agents to manage it.

- **DDS QoS:** The use of DDS protocol as the basis for MSCP allows to use certain QoS which are very convenient for its operation; the reliability QoS guarantees that every message sent is received by the corresponding destination entity, and the durability QoS that a message is kept for as long as it is required.
- **Node independence:** Each node is independent of the rest of the network, and it will behave exactly the same way in any scenario, which makes it very versatile and easy to implement. Also, the architecture does not depend on the task the server processes, what makes it very useful for many different assignments, and independent of the time in which each of them needs to be executed.

It is important to make clear that the MSCP is a communication mechanism that is in an early stage of development, and thus has not encountered any big issues so far. Still, it presents a number of limitations that may need to be addressed in later development phases. These are some of the problems that an architecture using MSCP could encounter:

- **Client drop:** When a client drops, the Server will not notice it, and will continue working on the received task even if its result no longer serves a purpose.
- **Data distribution:** Large data files (tasks and solutions) should only be sent from a client to its target server. However, this requires specific DDS features that were not available at the time of writing the present document.
- **Internal Synchronization:** This architecture is difficult to implement inside a Node, as it requires to synchronize several readers at the same time, and, in most cases, synchronize them from many threads. This leads to the challenging internal synchronization of an entity.

In future versions of this communication protocol, an effort will be made to increase the client-server feedback as well as public status information with the objective of improving the synchronization and efficiency of the architecture.

5.3.5 Topics and data types

In order to accomplish the MSCP communication, certain DDS topics and data types are required, even in the generic scenario (without knowing the specific task). It is important to notice that the MSCP is a generic protocol that does not depend on the underlying task that is being communicated. Thus, the topics and data types must be generic and reusable among tasks. One may think of a *task* as the kind of problem a server executes (e.g. in the case of AML-IP-Main and AML-IP-Computing the task is to obtain a new atomization from a batch of data).

As can be seen in figure 21, there exist 5 DDS topics (for each task). However, the number of types is lower because some topics may reuse others' types.

Data Types

- **Request Reference:** This data type uniquely identifies an availability request sent by a client to solve a particular task. Each client and task is identified by a unique id.
- **Task Reference:** This data type uniquely identifies a task that has already been assigned to a target server. It contains the client and server unique ids, as well as the id of the task.
- **Task:** This is the data type sent to the target server so it may execute the task. It references a specific task through a **TaskReference**, and contains the data to be processed. The nature of data depends on the task at hand, hence this data type must be generic enough so that, for any task under consideration, it is possible to expand this data type through class inheritance.

- **Solution:** This is the data type sent to the client after processing the task. It references a specific task through a **TaskReference**, and contains the solution for the processed task. This data also depends on the specific task, and hence must be implemented by means of inheritance.

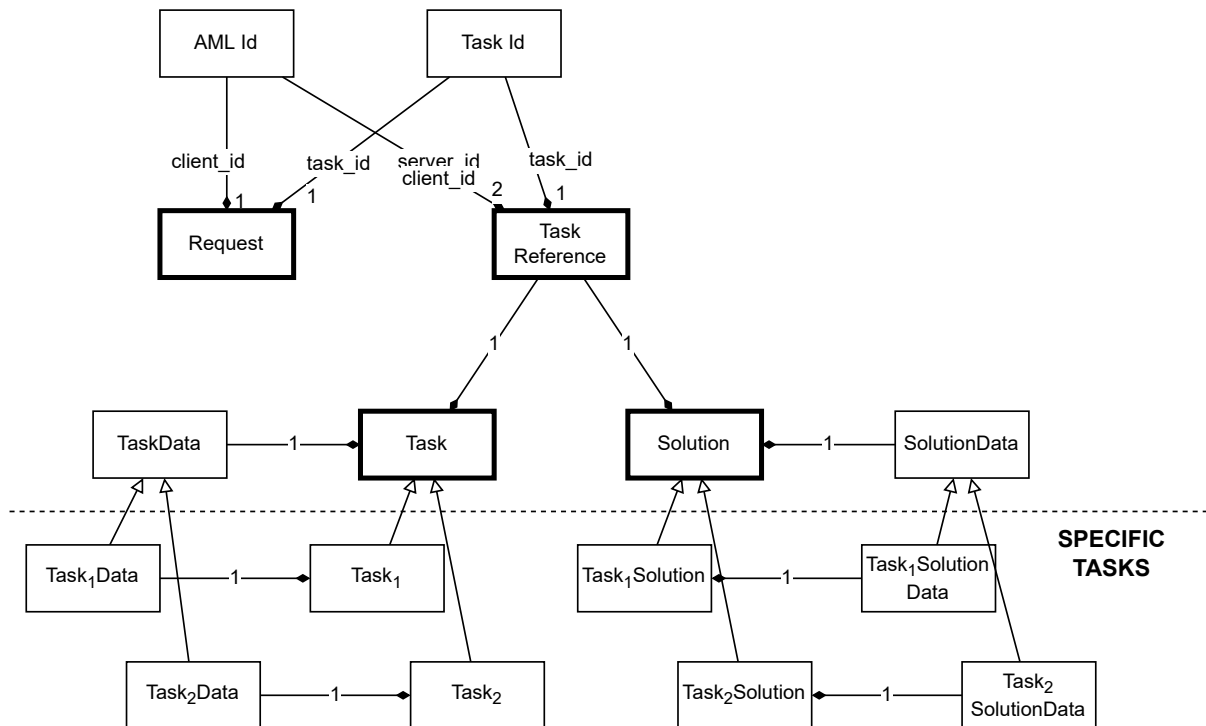


Figure 25: MSCP DDS Data Types.

Topics

- **request_availability:** client requests any server to notify their availability for a certain task.
- **reply_available:** server replies that is available for a specific client and task.
- **task_target:** client publishes which server is chosen to process that specific task.
- **task:** client publishes the task data to the target server.
- **solution:** server publishes the solution data for the given task to the source client.

Topic Name	Data Type	Node Publish	Node Subscribe
request_availability	RequestReference	Client	Server
reply_available	TaskReference	Server	Client
task_target	TaskReference	Client	Server
task	Task	Client	Server
solution	Solution	Server	Client

5.3.6 Use cases

The MSCP may be used for any scenario in which a client needs to commend a specific task to external servers that are specialized in performing such task. Obviously, this protocol perfectly fits the *AML-IP*

workload distribution scenario, as it has been implemented mainly for this purpose. Nevertheless, there exist other possible scenarios within the AML-IP for which this protocol may (and will) be used:

- **Inference Request:** Certain nodes (edge) may be required to perform inference on some data for an AML model, but not have enough computation power to do so. In this scenario, the edge node could work as a client and send the data to be processed (*task data*), so that any node with a proper atomization could perform the task and return the inference result (*solution data*).
- **Model Request:** When multiple nodes are working on the same atomization (model) collaboratively, only one of them would be the *best model* and all of them would have that knowledge. A node joining this network may request the best performing version of the model, and only one node should answer in order to avoid the exchange of redundant information.
- **Context Request:** It is very likely that a context description of the AML problems will be required in order to make the collaborative learning scenario possible, so that any node would be able to know on which problem it is working. This type of request could be sent from any node to a specific *context broker* node that would know how to handle it.

5.3.7 Alternatives

There were several already existing communication protocols that were considered for this scenario, but it was finally decided to design a new one, as none of them was able to fulfill all the requirements:

- **Server-Client over DDS:** Certain DDS environments as ROS 2 implement their own communication interfaces to recreate a Server-Client protocol using a specific architecture of DDS topics. However, this protocol is only able to connect one client with a single server, and does not allow to dynamically connect new servers to the network, neither to maintain the independence between a client and its server.
- **Master/slave model:** This communication model fulfills many of the requirements, but still fails to make possible for a server (slave) to work with different clients (masters), as every slave belongs to only one master.
- **Remote Procedure Call (RPC):** There exists an implementation of the RPC protocol over DDS developed by eProxima that could have been used for this purpose. Yet, it could not be utilized for this case because, in this architecture, every server receives requests from every client, and all of them would process the same data, thus duplicating effort.
- **Load Balancer:** A *Load Balancer* node would meet most of the requirements for this protocol. However, it is important to decentralize the application, avoiding single points of failure that would make the network highly vulnerable.

6 AML-IP workload distribution scenario Prototype

In order to show the current state of the AML-IP, and to proof that the presented designs work efficiently and as expected, eProxima has implemented a prototype of the AML-IP for workload distributed scenarios. This prototype is hosted on Github and publicly available at [aml-ip-prototype](https://github.com/eProxima/aml-ip-prototype/tree/ALMA_AMLIP_D6.2)¹⁵.

¹⁵https://github.com/eProxima/aml-ip-prototype/tree/ALMA_AMLIP_D6.2

6.1 Technology used

This repository follows the usual architecture of ROS 2 packages or Fast DDS workspaces, and it is easy to install and execute.

6.1.1 AML-IP Algebraic Nodes

The AML-IP-Alg are under development by ALGEBRAIC, and are not yet ready for deployment. For this reason, this prototype uses mocked nodes. That is, nodes that have the exact same input and output that the actual ones will have, and whose internal mechanism is simplified, or even eliminated. As the main objective of this prototype is to prove the feasibility of the communication process, and not the AML itself, these mocks are enough for this purpose.

6.1.2 Fast DDS Python Bindings

In order to bind the ALGEBRAIC mock nodes with Fast-DDS, Fast DDS Python Bindings are used to build the DDS communication architecture. This includes the creation and management of the different DDS entities (DomainParticipants, DataWriters, DataReaders, etc.), the QoS that each of them uses, and the processes for writing and reading messages.

6.1.3 eProxima DDS Router

This prototype is independent from the DDS Router, as restricting communication to a single LAN is enough to prove the feasibility of the given design. However, an experiment testing this prototype within a WAN scenario is also provided, in which the DDS Router is in charge of enabling the communication between external nodes.

6.2 Design

This design does not intend to be robust nor represent the future implementation of these nodes, but to test the communication processes between nodes. Thus, the nodes implemented for this prototype are rather simple and restricted to the desired behaviour. Future nodes are intended to be very flexible and their behaviour will relay on higher level entities.

6.2.1 Prototype nodes

Figure 26 schematically displays how each kind of node behaves. These nodes communicate by following the MSCP explained in subsection 5.3.

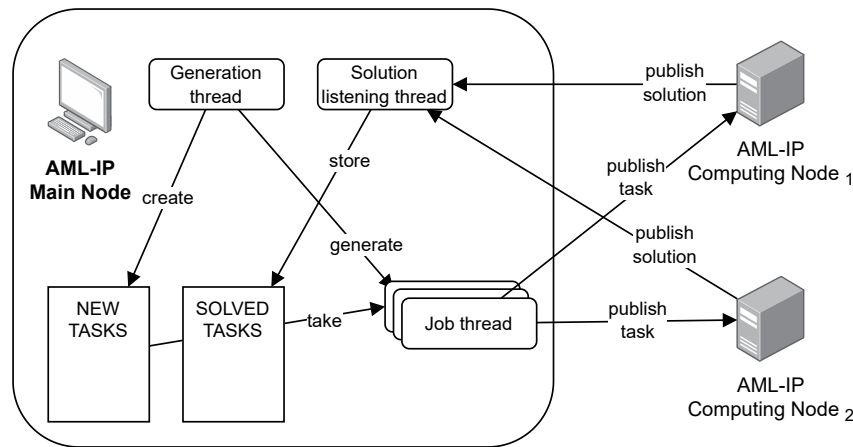


Figure 26: Prototype nodes design.

AML-IP-Main Each execution of this node has two main threads. The *Generation thread* creates new (random) tasks, each of them starting a new thread that is responsible for sending that corresponding data to a AML-IP-Computing and wait for the solution. Once the solution arrives, it stores it in a database. The *Solution listening thread* awakes with a fixed frequency. The mission of this thread is to join the threads that have already finished, and collect their solution to be stored in a shared database.

AML-IP-Computing The behaviour of the AML-IP-Computing nodes is much simpler. Each has a simple thread that awaits for a job targeted to them. Once the task arrives (after the required MSCP synchronization), the process waits for a random time, which simulates the training time, and returns the solution.

Data The data generated by the AML-IP-Main and the solution data retrieved by the AML-IP-Computing is transmitted as raw bytes (as will be the case for future AML data). An extra field has been added to the data being sent in order to have the possibility of verifying the correct operation of the whole system. This additional data entry consists of a string with a simple sentence randomly generated, whose end is completed (also randomly) by the AML-IP-Computing.

e.g.

<i>AML-IP-Main publish job</i>	<i>AML-IP-Computing publish solution</i>
<i>She was playing</i>	<i>She was playing alone.</i>
<i>Tim was talking</i>	<i>Tim was talking with me.</i>

Timing Both the time elapsed between job generations in AML-IP-Main and the time elapsed in a AML-IP-Computing since a task is received until it is resolved are random. This attempts to simulate the times that an actual node would take to process the data it is working with.

6.2.2 DDS Data types

The data types used in this prototype are the ones described in Figure 25. The IDL files used to generate those data types that Fast DDS Python Bindings requires can be found in appendix B.

6.3 Experiments

Different experiments and architectures have been tested to check the functionality of the prototype.

6.3.1 Simple experiment

This experiment follows a very simple approach where only one AML-IP-Main communicates with only one AML-IP-Computing, both displaying their expected behaviour.

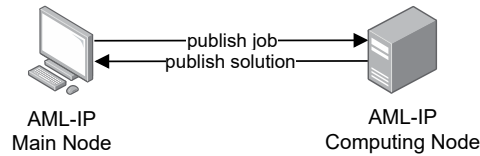


Figure 27: LAN prototype experiment architecture.

6.3.2 WAN experiment

In this experiment, the same simple architecture is used, but the two nodes are deployed in different LANs. This requires the use of the DDS Router (in the future AML-IP-Agent) to communicate all necessary data. This scenario is meant to demonstrate the use of an AML distributed network in WAN scenarios.

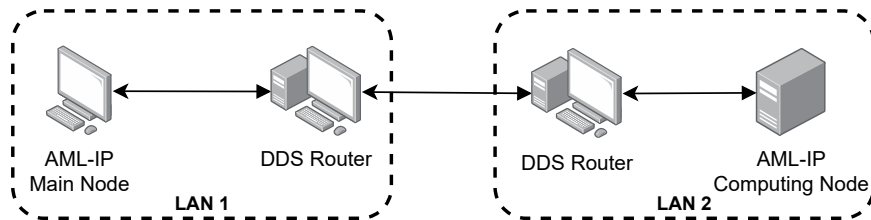


Figure 28: WAN prototype experiment architecture.

6.3.3 Complex experiment

This is the most relevant experiment, where the potential of DDS and MSCP synchronization is tested. Two (or an arbitrary number) AML-IP-Main nodes will work concurrently by making requests to a network of three (or an arbitrary number) AML-IP-Computing nodes. The workload will be dynamically divided between the AML-IP-Computing, that will process the requests neatly. This is the scenario that is demonstrated in subsection 6.4.

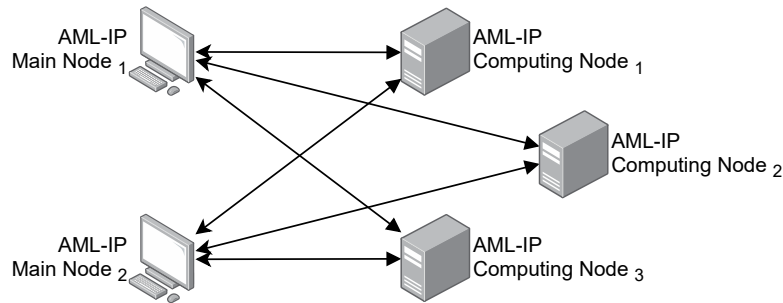


Figure 29: Complex prototype experiment architecture.

6.4 Demonstrator

The following section presents a demonstrator of the AML-IP prototype for the workload distribution scenario. 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 purpose of this demonstrator is to illustrate the successful connection between AML-IP prototype nodes parallelizing tasks in a local environment.

6.4.1 Experiment design

The implemented nodes are the AML-IP-Main and the AML-IP-Computing, which are the main components of the future AML-IP-Engine. Figure 29 shows the network architecture deployed for this demonstrator. This architecture is chosen to test the complex scenario where synchronization and multiple tasks must be managed by the nodes.

The expected behaviour of the prototype is that any request sent by the AML-IP-Main will be addressed by one and only one AML-IP-Computing after a certain period of time. The prototype prints logs and stores the results in final files so the execution of the whole system can be checked. Be aware that the time periods are random, and so it may occur that the AML-IP-Computing nodes do not answer quickly enough to the tasks handed by the AML-IP-Mains, resulting in a queue of non-resolved tasks. These tasks are addressed in the same order they were emitted, so eventually the system would be able to auto-adjust.

WAN This same scenario could be replicated in WAN with any distribution of the nodes across the LANs. It is only required to deploy an instance of DDS Router in each of the LANs, using configurations 6 and 7.

6.4.2 Download and installation

The prototype has been developed as an Open Source project in GitHub. Thus, the whole code and instructions are made available in the `aml-ip-prototype` GitHub repository¹⁶. The project has been developed for `Ubuntu 20.04.2.0 LTS (Focal Fossa)`¹⁷ OS and it has not been tested in any other platforms. However, every eProsima product is tested in *Windows* and *macOS* platforms, so this prototype may also be supported for these OSs.

¹⁶https://github.com/eProsima/aml-ip-prototype/tree/ALMA_AMLIP_D6.2

¹⁷<https://releases.ubuntu.com/20.04/>

The project `README.md` file contains 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`¹⁸ tool to build the project.

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/ALMA_AMLIP_D6.2/amlip.repos
$ vcs import src < amlip.repos
$ colcon build
```

6.4.3 Experiment setup

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 must be preceded by the following instructions in order to correctly configure the environment:

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

The commands required to launch each kind of node are:

```
$ ./install/amlip_nodes/lib/amlip_nodes/amlip_computing
$ ./install/amlip_nodes/lib/amlip_nodes/amlip_main
```

Both AML-IP-Main and AML-IP-Computing accept configuration arguments when executing the previous commands. Using the `-h` or `--help` argument, information on its usage and possible arguments is displayed.

- **–domain** Set the Domain Id for the DDS communication. This is useful to simulate the WAN scenario in a single computer.
- **–seed** Set the initial random seed to a specific number to make the experiment deterministic.
- **–debug** Activate debug traces.

6.4.4 Results

This experiment has been performed using the implemented prototype and following the instructions already given. In Appendix C, several screenshots taken while executing the commands from Section 6.4.3 are shown.

Understanding log traces Once everything is deployed, all nodes will discover each other dynamically using DDS multicast discovery. It could be seen in screenshots 36 and 37 when these nodes has started and how the AML-IP-Main has sent `job 0` to AML-IP-Computing, that has answered (approximately in the time that it takes to generate one or two other jobs) and has answered it. From that point, the rest of the nodes start to affect the network, and then the jobs from any AML-IP-Main are automatically distributed to first AML-IP-Computing that frees from other job.

¹⁸<https://colcon.readthedocs.io/en/released/>

Traces from AML-IP-Computing basically log the jobs that it starts to process, and also when the job solution is retrieved.

Traces from AML-IP-Main log each new generated job, announced via a new thread that sends a request and waits for an available server. It also logs every 5 seconds those jobs that have been resolved together with their solution.

Understanding final log files Each node generates a final log file in folder `./INTERCHANGE_FOLDER/RESULTS` with the name of the node as part of each file name. AML-IP-Computing generates a file with every job received and the solution found for it. AML-IP-Main generates a file that collects the jobs sent that have not yet been solved, as well as the resolved jobs together with their solution. Some examples of log files for AML-IP-Computing and AML-IP-Main can be seen in Listing 5 and Listing 8, respectively.

Duplicated task ids Each task is uniquely identified by two values, the id of the client and the id of the task. There are several repeated task numbers along the final log files, as there were two AML-IP-Main generating jobs, and both started numbering from 0. Checking the logs, it is easy to determine which AML-IP-Main any task belongs to just by inspecting the message. This is the main reason why these sentences are generated and answered randomly.

6.4.5 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:

- The synchronization and orchestration of many nodes working in the same training process does not require a centric application, and could be achieved easily and efficiently using DDS communication with higher abstraction layers as the MSCP.
- Working across different LANs while using the DDS Router results in an architecture equivalent to having all nodes in the same LAN (disregarding the obvious increase in latency affecting communication).
- The most complex part of the design of the prototype has certainly been the synchronization of the internal threads of each node. Most of the time, simultaneous threads must notify each other and also write or read from the same data structures. More subtle designs should be considered in order to reduce this complexity.
- The MSCP requires further thought in its design in order to handle drops and desynchronizations, as the current version of the prototype does not cover the situation where the connection fails once the server has been targeted. Both Main and Computing will get stucked waiting for the other nodes to respond.

7 Conclusions

This document presented the design of the AML-IP. It is not intended to be the final design but a sketch on how it is going to be built, and it could change as the project advances depending on the necessities that may 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.

7.1 Advantages

The AML-IP communication platform described in this document addresses all requirements needed for the correct and efficient execution of an AML environment.

Reliable software tools This architecture is based on several already available tested products, which facilitates the implementation and reduces the number of errors that could appear. Moreover, all these products are released under the Apache 2.0 Open Source license, and are receiving technical support by *EPROS*.

The development of the communication platform will have the AML environment as main focus, 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 Centered Design, i.e. the entire user experience with the communications framework has been taken into consideration. Thus, AML users could leverage the AML-IP communication benefits without the need of knowing, configuring or having to 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 a 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 fixed 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 participating 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-Agents is only forwarded to the nodes that require 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 when compared to other P2P possible solutions as Block Chain.

Security The security of communications when working in the Internet is crucial, so it is a matter to be taken into account in every communications framework. 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 data encryption over the Internet is given by *Fast DDS* communications middleware, avoiding the need for extra security features. There exists the possibility of increasing point to point communication security by using TLS over TCP, but it may not be a desired feature as the AML-IP is designed to be an open environment where any new node could enter the network at any point and receive the whole information. The possibility of adding a logging or authentication mechanism to join a network of AML-IP-D nodes will be considered for future versions of the AML-IP.

7.2 Limitations

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

Centralized communication from LAN to WAN The AML-IP-Agent centralizes the connection between the user private network and the Internet, by creating a secure, independent WAN communication that does not affect 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 would be 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 needs 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 purposes, as the whole data needs to be sent from one AML-IP-Alg to the rest of the network in a reliable way and with a frequency higher than that with which new data is generated.

The first approach considered to address this problem would be to fragment the data to be then sent reliably. Another option considered would be that of reducing the amount of data required by an AML node. Thus, not all the data generated by an AML node would be sent, but only an update with respect to previously sent data. The possible features of this functionality will be discussed and agreed with ALG and the AML development team.

Data privacy mixing scenarios There are two principal scenarios where, by using the AML-IP, two of the main strengths of the AML algorithm are exploited: the *collaborative learning* and *workload distribution* scenarios. These scenarios are not mutually exclusive. Nodes are designed to work in one specific scenario, but any machine or process could have different nodes from different scenarios working together. There is however a problem encountered when combining the two scenarios.

Data privacy when training a model is a main concern for AML, and the data used is kept private within the collaborative learning scenario. However, if a node requests the use of remote nodes for computational purposes (workload distribution), the privacy of data exchanged in this process can no longer be assured, and may be difficult to do so.

Several solutions are being considered to solve this issue, the main ones relying on the encryption of data in the communication phase, or even in the training one, and to decrypt the solution in a later step.

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

```
eProxima@eProxima-XPS-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-XPS-15-7590:~/AML$
```

Figure 31: AML-IP-D 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 32: AML-IP-D 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 33: AML-IP-Engine in HOST 2.

```

AML_IP_DiscoveryServer
paris@paris-XPS-15-9570:~/projects/AML-IPS ./install/amlip/bin/AML_IP_DiscoveryServer --listening-addresses="127.0.0.1,5002;37.35.177.91,5000" --connection-addresses="37.11.18.30,5000,1" --time 0 --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: 5000
connecting with servers in addresses:
IP: 37.11.18.30 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.8d.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.8d.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.8d.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.8d.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-IPS

```

Figure 34: AML-IP-D in HOST 3.

```

amlip DL
paris@paris-XPS-15-9570:~/projects/AML-IPS ./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.8d.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: 10>
paris@paris-XPS-15-9570:~/projects/AML-IPS

```

Figure 35: AML-IP-DL in HOST 3.

B IDLs for workload distribution scenario prototype

```

1
2 // Uniquely identifies a request from a client
3 struct Multiservice_RequestAvailability
4 {
5     string requester_id;
6     unsigned long task_id;
7 };
8
9 // Uniquely identifies a processing task by the client , the task and the
   target server
10 struct Multiservice_TaskReference
11 {
12     string requester_id;
13     unsigned long task_id;
14     string server_id;
15 };
16
17 // Request Data Type for the specific task
18 struct Multiservice_Task_Data
19 {
20     // To inherit in specific task data type
21 };
22
23 // Identifies the task processing and adds the data (that depends on the
   specific task)
24 struct Multiservice_Task
25 {
26     Multiservice_TaskReference task_reference;
27     // This must be implemented in the subclass
28     // Multiservice_Task_Data data;
29 };
30
31 // Solution Data Type for the specific task
32 struct Multiservice_Solution_Data
33 {
34     // To inherit in specific task data type
35 };
36
37 // Identifies the task processing and adds the solution data (that depends
   on the specific task)
38 struct Multiservice_Solution
39 {
40     Multiservice_TaskReference task_reference;
41     // This must be implemented in the subclass
42     // Multiservice_Solution_Data data;
43 };

```

Listing 3: MultiService Communication Protocol generic Data Types.

```

1
2 // Specific Data Type for a Job-task request
3 struct Job_Task_Data : Multiservice_Task_Data
4 {
5     // Raw data in bytes
6     sequence<octet> job;
7 };
8
9 // Specific Data Type for a Job-task solution
10 struct Job_Solution_Data : Multiservice_Solution_Data
11 {
12     // Raw data in bytes
13     sequence<octet> solution;
14 };
15
16 // Inherit Data Type for a Job-task
17 struct Job_Task : Multiservice_Task
18 {
19     Job_Task_Data data;
20 };
21
22 // Inherit Data Type for a Job-task solution
23 struct Job_Solution : Multiservice_Solution
24 {
25     Job_Solution_Data data;
26 };

```

Listing 4: MultiService Communication Protocol specific Data Types for Job.

C Screenshots of Demonstrator for workload distribution results

```
allowlist:
  # each of the AML-IP MSCP topics required
  - name: aml/status
    type: aml::status
  - name: aml/ms/request_availability/job
    type: aml::ms::request
  - name: aml/ms/reply_available/job
    type: aml::ms::task_reference
  - name: aml/ms/task_target/job,
    type: aml::ms::task_reference
  - name: aml/ms/task/job
    type: aml::ms::task::job
  - name: aml/ms/solution/job
    type: aml::ms::solution::job

simple_0:      # Simple Participant to communicate locally in Domain Id 0
  type: local
  domain: 0

wan:          # External Participant
  type: wan
  id: 0
  listening-addresses:
    - ip: "x.x.x.x"      # This machines IP
      port: 11666        # This machine open port to Internet
```

Listing 6: eProxima DDS Router YAML configuration for WAN communication as a Server.

```
paris@paris-XPS-15-9570:~/projects/amlip$ ./install/amlip_nodes/lib/amlip_nodes/amlip_computing
2022-02-24 15:41:02,966 - USER - Starting AML-IP Computing Node 049 execution.
2022-02-24 15:41:06,568 - USER - AML-IP Computing Node 049 calculating result for job 0 (approximately 3.305 ms).
2022-02-24 15:41:09,877 - USER - AML-IP Computing Node 049 finish calculating job 0.
2022-02-24 15:41:15,280 - USER - AML-IP Computing Node 049 calculating result for job 1 (approximately 2.8 ms).
2022-02-24 15:41:18,084 - USER - AML-IP Computing Node 049 finish calculating job 1.
2022-02-24 15:41:18,580 - USER - AML-IP Computing Node 049 calculating result for job 2 (approximately 4.378 ms).
2022-02-24 15:41:22,961 - USER - AML-IP Computing Node 049 finish calculating job 2.
2022-02-24 15:41:22,965 - USER - AML-IP Computing Node 049 calculating result for job 4 (approximately 6.692 ms).
2022-02-24 15:41:29,661 - USER - AML-IP Computing Node 049 finish calculating job 4.
2022-02-24 15:41:29,664 - USER - AML-IP Computing Node 049 calculating result for job 5 (approximately 3.328 ms).
2022-02-24 15:41:32,996 - USER - AML-IP Computing Node 049 finish calculating job 5.
2022-02-24 15:41:34,482 - USER - AML-IP Computing Node 049 calculating result for job 7 (approximately 4.136 ms).
2022-02-24 15:41:38,623 - USER - AML-IP Computing Node 049 finish calculating job 7.
^C2022-02-24 15:41:45,291 - USER - Signal handler called with signal 2.
2022-02-24 15:41:45,291 - USER - Signal received, stopping node AML-IP Computing Node 049. This could take few seconds.
2022-02-24 15:41:45,292 - USER - Finishing AML-IP Computing Node 049 execution.
paris@paris-XPS-15-9570:~/projects/amlip$
```

Figure 36: Prototype AML-IP Algebraic Computing Node execution traces.

```

allowlist:
  # each of the AML-IP MSCP topics required
  - name: aml/status
    type: aml::status
  - name: aml/ms/request_availability/job
    type: aml::ms::request
  - name: aml/ms/reply_available/job
    type: aml::ms::task_reference
  - name: aml/ms/task_target/job,
    type: aml::ms::task_reference
  - name: aml/ms/task/job
    type: aml::ms::task::job
  - name: aml/ms/solution/job
    type: aml::ms::solution::job

simple_1:      # Simple Participant to communicate locally in Domain Id 1
  type: local
  domain: 1    # Another non default domain is used in case is proven in same LAN

wan:          # External Participant
  type: wan
  id: 1        # This Router id
  connection-addresses:
    - id: 0      # Remote Router id
      addresses:
        - ip: "x.x.x.x"      # Remote machines IP
          port: 11666        # Remote machine open port

```

Listing 7: eProsimas DDS Router YAML configuration for WAN communication as a Client.

```

2022-02-24 15:41:04,345 - USER - Starting AML-IP Main Node 382 execution.
2022-02-24 15:41:06,566 - USER - Main Node AML-IP Main Node 382 has created a new job:
  JOB: {0: <She is bubbling >}}

2022-02-24 15:41:11,383 - USER - Main Node AML-IP Main Node 382 has created a new job:
  JOB: {1: <He was bubbling >}}

2022-02-24 15:41:11,571 - USER - Main Node AML-IP Main Node 382 has received the solution to job:
  JOB: {0: <She is bubbling >}}
  with result:
  SOLUTION: {0: <She is bubbling badly.>}}

2022-02-24 15:41:15,646 - USER - Main Node AML-IP Main Node 382 has created a new job:
  JOB: {2: <He is reading >}}

2022-02-24 15:41:18,778 - USER - Main Node AML-IP Main Node 382 has created a new job:
  JOB: {3: <Jack was bubbling >}}

2022-02-24 15:41:21,578 - USER - Main Node AML-IP Main Node 382 has received the solution to job:
  JOB: {1: <He was bubbling >}}
  with result:
  SOLUTION: {1: <He was bubbling like a boss.>}}

```

Figure 37: Prototype AML-IP Algebraic Main Node execution traces.

```

1 JOBS ANSWERED
2 {0: <She is bubbling >} : {0: <She is bubbling badly.>}
3 {1: <She was speaking >} : {1: <She was speaking with me.>}
4 {2: <Tim is reading >} : {2: <Tim is reading like a boss.>}
5 {4: <Tim was reading >} : {4: <Tim was reading like a boss.>}
6 {5: <Lorenzo was reading >} : {5: <Lorenzo was reading alone.>}
7 {7: <He likes speaking >} : {7: <He likes speaking very well.>}

```

Listing 5: AML-IP Algebraic Computing Node final log file from node AML-IPComputingNode049.

```

1 JOBS ANSWERED
2 {0: <Jack is bubbling >} : {0: <Jack is bubbling with me.>}
3 {3: <Jack was bubbling >} : {3: <Jack was bubbling very well.>}
4 {3: <He is reading >} : {3: <He is reading like a boss.>}
5 {4: <Tim was bubbling >} : {4: <Tim was bubbling like a boss.>}
6 {6: <Tim is bubbling >} : {6: <Tim is bubbling like a boss.>}
7 {7: <She was speaking >} : {7: <She was speaking like a boss.>}
8 {8: <She likes reading >} : {8: <She likes reading like a boss.>}

```

Listing 6: AML-IP Algebraic Computing Node final log file from node AML-IPComputingNode263.

```

1 JOBS ANSWERED
2 {1: <He was bubbling >} : {1: <He was bubbling like a boss.>}
3 {2: <He is reading >} : {2: <He is reading like a boss.>}
4 {5: <He likes talking >} : {5: <He likes talking alone.>}
5 {6: <Tim is talking >} : {6: <Tim is talking badly.>}

```

Listing 7: AML-IP Algebraic Computing Node final log file from node AML-IPComputingNode977.

```

1 PENDING
2 {8: <She was playing >}
3 SOLUTION
4 {0: <She is bubbling >} : {0: <She is bubbling badly.>}
5 {1: <He was bubbling >} : {1: <He was bubbling like a boss.>}
6 {3: <Jack was bubbling >} : {3: <Jack was bubbling very well.>}
7 {2: <He is reading >} : {2: <He is reading like a boss.>}
8 {4: <Tim was reading >} : {4: <Tim was reading like a boss.>}
9 {7: <She was speaking >} : {7: <She was speaking like a boss.>}
10 {6: <Tim is bubbling >} : {6: <Tim is bubbling like a boss.>}
11 {5: <He likes talking >} : {5: <He likes talking alone.>}

```

Listing 8: AML-IP Algebraic Main Node final log file from node AML-IPMainNode382.

```

1 SOLUTION
2 {0: <Jack is bubbling >} : {0: <Jack is bubbling with me.>}
3 {2: <Tim is reading >} : {2: <Tim is reading like a boss.>}
4 {1: <She was speaking >} : {1: <She was speaking with me.>}
5 {3: <He is reading >} : {3: <He is reading like a boss.>}
6 {5: <Lorenzo was reading >} : {5: <Lorenzo was reading alone.>}
7 {4: <Tim was bubbling >} : {4: <Tim was bubbling like a boss.>}
8 {8: <She likes reading >} : {8: <She likes reading like a boss.>}
9 {7: <He likes speaking >} : {7: <He likes speaking very well.>}
10 {6: <Tim is talking >} : {6: <Tim is talking badly.>}

```

Listing 9: AML-IP Algebraic Computing Node final log file from node AML-IPMainNode413.

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>.