Real Time Distributed and Decentralized Peer-to-Peer Protocol for Swarm Robots

Mahmoud Almostafa RABBAH[1], Nabila RABBAH[2], Hicham BELHADAOUI[3], Mounir RIFI[4]
RITM Laboratory, ESTC, Hassan II University, Casablanca, Morocco[1, 3, 4]
Laboratory of Complex Cyber Physical Systems, ENSAM, Hassan II University, Casablanca, Morocco[2]

***Abstract*—This contribution proposes an approach to enhance the capability of robotic agents to join the Internet of Things (IoT) and act autonomously in extreme and hostile environment. This capability will help in the development in environments where the connectivity, availability, and responsivity of the devices are subject to variations and noises. A real time distributed and decentralized Peer-to-Peer protocol was designed to allow Autonomous Unmanned Surface Vessels (AUSV) extend their context awareness. The developed Middleware allows a real time communication and is designed to run on top of a Real Time Operating System (RTOS). Furthermore, the proposed Middleware will give researchers access to a large amount of data collected by sensors, and thus solve one of the major problems encountered while training artificial intelligence models which is the lack of sufficient data.**

***Keywords*—Autonomous robots; smart objects; peer-to-peer; real time communication; ROS2; ZeroMQ; middleware**

I. Introduction

In the past, static robots, such as industrial arms, were used to perform repetitive tasks in a production line where the environment is well controlled and known in advance, at that time, collaboration between robots was not a priority. However, we are increasingly seeing the emergence of applications involving a swarm robot that share a common ultimate goal, e.g., The Autonomous Unmanned Surface Vessels (AUSV) or Unmanned Ground and Aerial Robots that must achieve missions like first responders, coast guards, area search, target detection and tracking, formations, rendezvous [1-3].

From these facts, the research on collaborative robots has increased considerably, and many researchers have started to make their focus on the internal design of the robot's context awareness [4-5], and the trend is to use Mobile Robots in hostile environments where the stability of the surrounding conditions and the connectivity is limited.

Mobile Robots require collaborative capabilities to achieve complex missions on hostile environment, e.g., AUSV may need to collaborate to build a mesh network where each AUSV serves as a network node. However, most of proposed AUSV was designed to operate in an already known environment and are not designed to adapt themselves to the new changes in the context.

We propose middleware for collaboration, communication, device hardening for deployments in extreme environments. We explore Multi Agent Systems (MAS) as a solution to enhance the collaboration by increasing autonomy, flexibility, and composability of robotic agents with the IoT devices available on their surrounding environment to promote the self-awareness of those agents. Not only the sensing and actuation are considered, but we also look at the distribution of decision-making in term of collaboration between the components of the application.

Our proposed Middleware named Collaborative Open Platform for Distributed Artificial Intelligence (COPDAI) allows a real time communication between a community of robots while supporting link and component degradation. The community takes distributed decisions that position agents on strategic locations to mitigate the risk of disconnection. Position depends also on the capabilities such as sensing and actuating. Agents are interconnected and they maintain this interconnection as principal vehicle of communication among them in a peer-to-peer mode.

Another problem that COPDAI will try to solve is the difficulty of having access to sufficient data to train artificial intelligence models, COPDAI will promote the sharing of sensor data and the trained models within the scientific community, as well as within the mobile robots.

II. Related Work

In recent study [6] authors presented multiple node communication mechanisms: Simple message, Ports, Topics, Events and services, and based on pre-established criteria, they compared several Robotics Software Framework (RSF) to evaluate the coverage of each of them to defined criteria. It is worth mentioning that robotic systems are often designed over an Ethernet. Field Buses, such as CANBus, I2C, EtherCAT, Serial lines, FireWire, PROFIBUS, and even PCI are often used. Unfortunately, most RSFs and MASs use only the IP protocol.

Generally, the MAS was used for its great flexibility and the ability to reuse components in different projects. Several patterns have been proposed for its implementation in Multi-Robot Systems, proving a gain in development time [7], in this Work Jade Middleware was used to ensure communication.

Agents distribution can be categorized into three forms [8]: Embedded agents at the robot level, agents located at a server level or hybrid distribution: Intelligence and computational agents are external to the robot, and acquisition and control agents are embedded.

In [9], the authors worked on the control of soccer robots, three schemes based on the multi-agent system paradigm were established: the first scheme is based on the control of the robots from a remote computer, in this configuration the robots had no embedded intelligence, the second scheme is based on a distributed architecture where the vision and the decision are done on a central computer and the control of the motors is delegated to embedded systems attached to the robots, the third scheme allows a greater autonomy of the robots where the acquisition of the sensor data, the decision as well as the control of the motors are done at the level of the robot, in addition to an eventual communication between robots.

In [10] the authors have proposed a distributed knowledge base, this base is shared between them. The agents are organized in a hierarchical way and in case of errors that occur in an agent belonging to the lower level, the agents of the higher level replan the trajectory of the robot.

In [11], the authors based their middleware on the Real-time CORBA specification [12] which extends the basic CORBA model to support real-time constructs. A client/server model was adopted, and the predictability improvement was based on Real-time CORBA mechanisms such as: thread-pooling and priority assignment.

In [13], the authors focused on the support of networking and middleware of mobile embedded systems, a communication protocol named TDMA allowed the transmission of data and manage the uncertainty related to the communication, in addition a shared memory named RTDB was defined to allow the agents to share data.

The authors in [14] developed a Humanoid Robot using XBotCore middleware, for real time communication, the middleware use EtherCAT protocol, and the software was built on the top of Xenomai RTOS, the middleware was designed to satisfy 1 kHz control frequency and implement four tasks in real time behavior among other: robot kinematic chain, robot joints, robot Force/Torque sensors. In [15], a middleware based on the concept of control kernel has been developed. Different types of nodes have been designed on top of two protocols namely: CAN bus and Ethernet, the nodes have different capabilities and can provide different types of services depending on their computing power. Lightweight nodes communicate on top of CAN bus and powerful nodes on top of Ethernet.

Also, in [16] we studied 14 Middlewares which are either oriented to robotics applications or smart objects applications, we concluded that most of the Middlewares do not meet the real time constraint like: UBIWARE [17], LMAARS [18], ACOSO [19], Voyager [20], JCAF [21], Aura [22], UBIWARE [23], LMAARS [24] and SOCRADES [25], while others suffer from a centralized architecture like ROS [26], ICARS [27], COROS [28].

III. COPDAI Communication Architecture

This Each sensor, actuator or decision module can be attached to the robot body or located in its external environment; we will represent each of these components by a node.

Due to the constraint of the hostile environment, our architecture must be robust to the instability of the physical communication links, thus each node can appear and disappear at any time, the Middleware must allow each node to detect the presence of the other nodes and must implement a recovery mechanism in case of communication failure.

In addition to that, our architecture must not have a Single Point of Failure (SPOF): the degradation of a node must not compromise the whole robot's mission, or at least we must be able to switch to a safe position, for that the architecture must be decentralized, we propose a Peer-to-Peer communication between the nodes.

Also, we need to allow distributed computing between nodes: thus, a node that is located on a computer/server with more resources (CPU, RAM…) can contribute to the computations that a node located on an embedded board with limited resources cannot do by itself.

In addition to that, the constraint of real time requires us to define a priority between the transmitted messages, and thus allow the node to process these messages with a minimum level of guarantee and a predictable behavior.

Finally, the middleware must promote collaboration within the scientific community through the sharing of content and collected data during experiments (sensor data, actuators data…) and optionally results or the trained model.

We distinguish four families of possible communication between these nodes among others (Fig. 1):

- Inter-robot communication:
  - Communication between nodes located in the same embedded card / computer.
  - Communication between nodes located in separate embedded cards / computers.
- Communication between robots.
- Communication between robots and smart objects.
- Content sharing (images, videos…) between researchers / robots.

During the design of the communication layer of COPDAI Middleware, we had to provide answers to the following functional requirements:

- Discovery: How can the nodes recognize each other, knowing that they can be located on the same embedded board or on remote embedded boards?
- Presence: How do we track the appearance and disappearance of nodes? Are we going to use a central component as advocated by multi-agent systems or are we going to use a distributed mechanism with partial knowledge of the topology?
- Connectivity: How do we connect one node to another? Are we going to use ethernet communication (on the same segment or on different network segments) or are we going to use inter-process communication (IPC)?

- Point-to-Point messaging: How to send a message from one node to another? Using a central system such as a message broker, or direct communication?
- Group messaging: How we can do group messaging? Use push/pull pattern or use publish/subscribe pattern?
- Real Time communication: How to prioritize critical messages and ensure that they are processed in real time?
- Content distribution: How to send the data collected by the robot embedded system (sensor data, execution data or engine logs...)? Are we going to use a decentralized protocol like (FileMQ [29], IPFS [30] ...)? Or are we going to use server-centric protocols like (FTP, HTTP...)?
- Bridging: How we can do wide area bridging?
- Security: How nodes protect the information they carry? And how to secure messages and content during the exchange operation?
- Test & Simulation: How do we simulate large numbers of nodes? Are we going use real embedded systems? Or are we going provide a way to do a software simulation?
- Distributed logging: What strategy to adopt to trace communications and collect logs from the nodes in order to detect possible failures or to debug?

A. *Transport Layer*

We choose the concurrency framework ZeroMQ [31] as transport layer, it gives us sockets that carry atomic messages across various transports, among others: IPC and TCP, researchers evaluate the performance of OpenDDS, ORTE and ZeroMQ middleware in terms of latency and scalability, they choose the publish/subscribe pattern to study those middleware performances and results show that ZeroMQ has the best performance with minimal latency [32]. Also, researchers here [33-34] have found that ZeroMQ scales much better and can smoothly handle high data loads and even bursts of requests, which was not the case in their old middleware version based on CORBA.

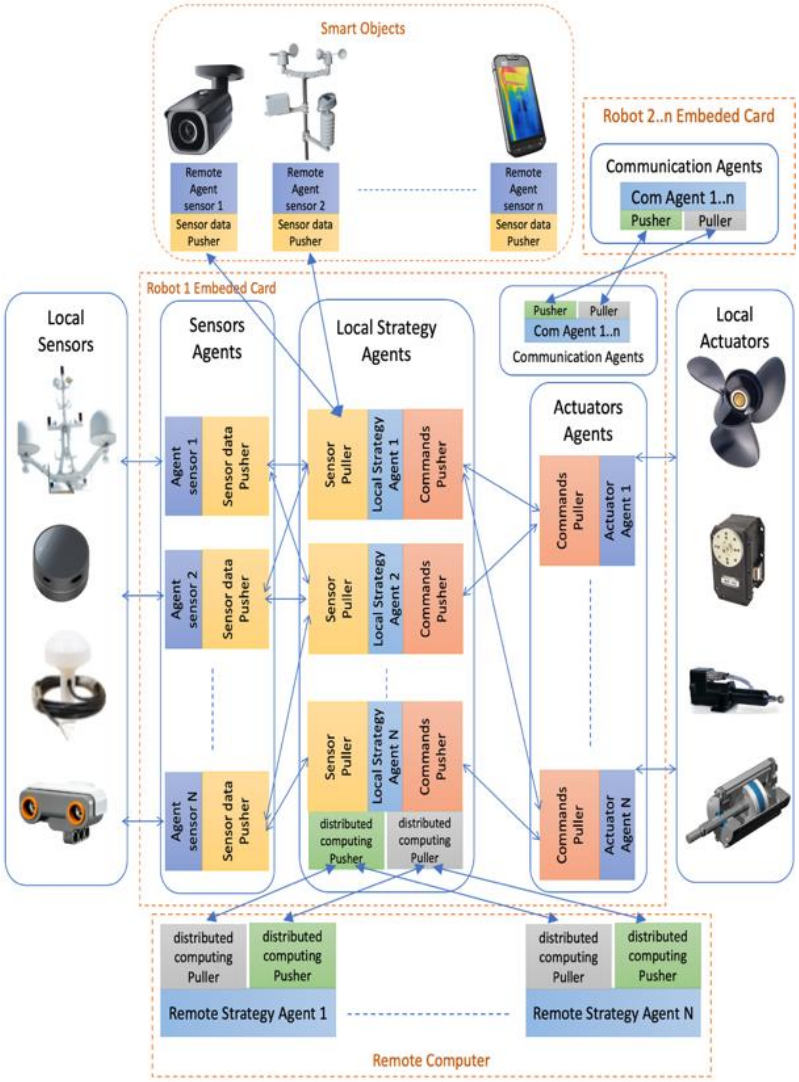

Fig. 1. Communication Families between COPDAI Nodes.

B. Transport Mechanisms

COPDAI will support in its first version the following transport mechanisms: TCP/IP, UDP/IP, and IPC, other mechanisms will be supported in the future releases such as Bluetooth, Serial wire and Acoustic communications.

Nodes within the same embedded card/computer will use IPC to communicate with each other and nodes located on different embedded boards/computers will communicate using IP protocols.

We were inspired by the ZeroMQ Realtime Exchange Protocol (ZRE) which governs how a group of peers on a network discover each other, organize into groups, and send each other events [35]. ZRE runs over the ZeroMQ Message Transfer Protocol (ZMTP). ZRE has been designed to run in smart home and can accept a limited number of nodes: each node establishes a connection to the other ones, which means if we have N nodes we are going to have $\frac{N \times (N-1)}{2}$ connections, which can cause the saturation of a network quickly. Another problem is that ZRE support only IP communication which represents an unjustifiable overhead in our case for the nodes that must communicate within the same embedded card / computer. And finally, ZRE has not implemented any notion of service.

COPDAI supports 4 Messaging types:

- Node to Node messaging: Nodes that belong to the same hierarchical group, and that are located on the same physical medium (embedded card / network segment) can communicate directly in Peer-to-Peer.
- Topic messaging: it is the case where some nodes want share message about the same topic.
- Hierarchical messaging: Nodes are organized in groups that accept a maximum number k of members, each group contains a leader, communication between members of the same group is direct, but communication between two nodes belonging to different groups must go through the respective leaders of each group,
- Bridging messaging: Communication between nodes belonging to two physical boundaries (two embedded cards or two Network segments) passes through a dedicated node, this node is elected among the group leaders.

Fig. 2 shows a use case of communication types with k=3, if we compare ZRE with COPDAI in this use case, in Network Segment 1 we have only 3 IP connections instead of 171 using ZRE, also ZRE does not allow communication between nodes in segment 1 and 2:

C. Discovery on the Same Machine

In a specific folder location, within the user home directory, each node creates a file with its UUID as file name. Each $\Delta_t - \varepsilon$ the node modifies its file timestamp.

Each Δ_t nodes list the files whose last modification date is less than Δ_t, and so, they will be able to know the new nodes that have just appeared or those that have disappeared (Fig. 3).

D. Discovery over IP

We want to keep back compatibility with the ZRE protocol for discovery over IP Protocol, so we are going to use the same mechanism: ZRE uses UDP IPv4 beacon broadcasts to discover nodes. Each ZRE node shall listen to the ZRE discovery service which is UDP port 5670. Each ZRE node SHALL broadcast, at regular intervals, on UDP port 5670 a beacon that identifies itself to any listening nodes on the network [35].

The header shall consist of the letters ‘Z’, ‘R’, and ‘E’, followed by the beacon version number, which shall be %x01.

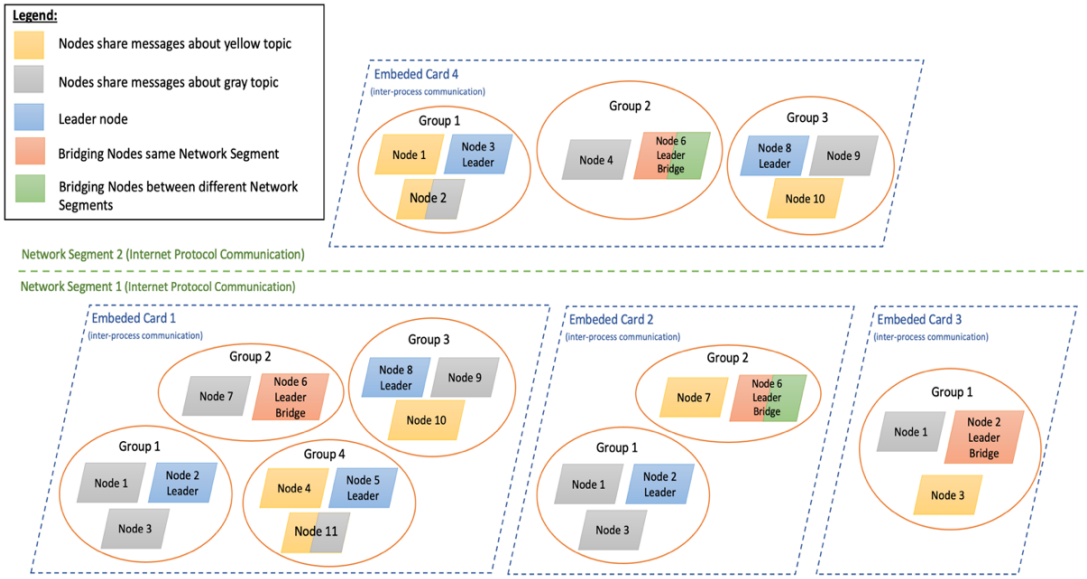

Fig. 2. COPDAI Communication Types with Groups that Accept at Maximum 3 Members.

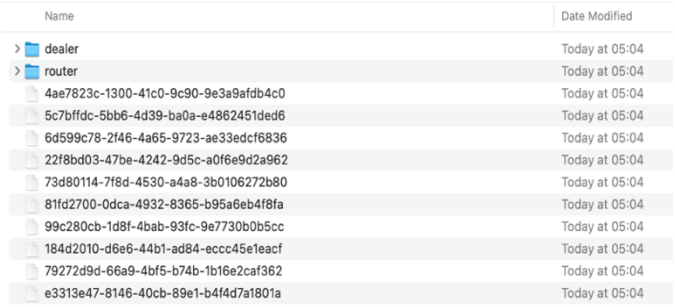

Fig. 3. IPC Discovery Files.

The body shall consist of the sender's 16-octet UUID, followed by a two-byte mailbox port number in network order. If the port is non-zero this signals that the peer will accept ZeroMQ TCP connections on that port number. If the port is zero, this signals that the peer is disconnecting from the network. The body contains also another two-byte mailbox port number for real time communication channel, and since in our case the Bridge node hides behind it several nodes which should be discoverable to the outside world, we will extend the ZRE beacon so that the body contains UUIDs of these nodes (Fig. 4).

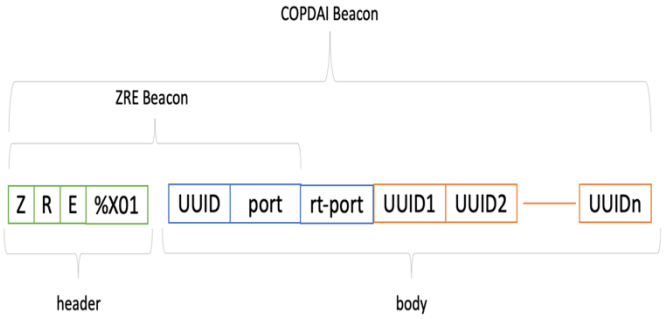

Fig. 4. COPDAI Beacon Message.

Node that receives a valid beacon with a non-zero port number will be considered as a new peer.

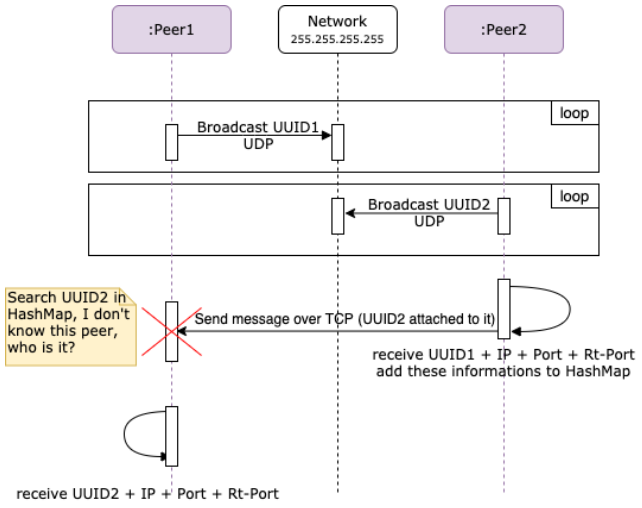

Fig. 5. Use Case where we can receive a Message before Receiving Beacon from Peer.

UDP messages are limited to 1500 bytes on LANs and 512 bytes on Internet, so the bridge node cannot handle more than 92 nodes in LANs and 30 nodes in Internet, if the bridge node reaches its limit, a new one is elected and handle the rest of the nodes.

Another problem is that bridge node can get the first beacon from a peer after it starts to receive messages from it, so in this situation we got a message from a node that we don't know its IP address and port (Fig. 5).

So, we must consider discovery over TCP: Our first command to any new peer to which we connect is an "Hello" command with our IP address and ports. Bellow the steps we will follow:

- If we receive a UDP beacon from a new peer, we connect to the peer through a TCP socket.
- Each message must contain the UUID of the sender.
- If it's a Hello message, we connect back to that peer if not already connected to it.
- If it's any other message, we must already be connected to the peer, if it is not the case, we raise an assertion.
- We send messages to each peer using the per-peer socket, which must be connected.
- When we connect to a peer, we also tell our Node that the peer exists.
- Every time we get a message from a peer, we treat that as a heartbeat.

Fig. 6 shows the message format for the "Hello" command throw IP.

```
hello           = signature %d1 version sequence endpoint groups status name headers
signature       = %xAA %xA2             ; two octets
version         = number-1              ; Version number (2)
sequence        = number-2              ; Cyclic sequence number
endpoint        = string                ; Sender connect endpoint
rtendpoint      = string                ; Sender real time endpoint
uuids_services  = strings               ; list of nodes under the sender's responsibility with svc
groups          = strings               ; List of groups sender is in
status          = number-1              ; Sender groups status value
services        = strings               ; List of services offered by the sender
name            = string                ; Sender public name
headers         = dictionary            ; Sender header properties
```

Fig. 6. COPDAI Hello Message throw IP.

Bellow we explain the signification of each part of the "Hello" Message:

1) Part 1: It is an Event Type (4 bytes), it is equal to %d1,

2) Part 2: It is the signature which let us control the received message is a COPDAI Message, must always equal to %xAAA2,

3) Part 3: It is the protocol version.

4) Part 4: It is a sequence number which will allow our node to check if there were any lost messages between the current received message and the last received one, for each peer.

5) Part 5: It is a string that concatenates the IP address of the peer and its port, the endpoint is specified as "tcp://ipaddress:mailbox".

6) Part 6: It is a string that concatenates the IP address of the peer and its real time port, the rtendpoint is specified as "tcp://ipaddress:rt-mailbox".

7) Part 7: list of UUIDs nodes under the responsibility of the sender and for each UUID list of proposed services.

8) Part 8: List of groups to which the peer belongs.

9) Part 9: The "group status sequence" is a one-octet number that is incremented each time the peer joins or leaves a group. Each peer may use this to assert the accuracy of its own group management information.

*10)*Part 10: List of services offered by the sender.

*11)*Part 11: A Human friendly peer's name.

*12)*Part 12: Headers is a hash table (Key/Value HashMap) of additional information that the peer can eventually send.

E. Detecting Disappearances over IP

Several reasons can come into play and distort the decision that a peer has really disappeared: due to high TCP traffic the UDP packets can be dropped (which causes a high latency before getting the beacon) or a high latency before getting a message on top of the TCP and which is also considered as heartbeat.

To overcome this problem, if we don't get a beacon from the peer after a while, we switch to TCP heartbeats which consist of sending a PING command and receiving a PING_OK response, the PING command is described in ZRE protocol as follow (Fig. 7).

```
ping            = signature %d6 version sequence
version         = number-1              ; Version number (2)
sequence        = number-2              ; Cyclic sequence number
```

Fig. 7. PING Command Sent to a Peer if it Disappears [35].

Bellow we explain the signification of the new part of the "PING" Message:

- Part 1: It is an Event Type (4 bytes), t is equal to %d6.

If the Peer is still alive it must respond with a PING_OK as described in Fig. 8:

```
ping_ok         = signature %d7 version sequence
version         = number-1              ; Version number (2)
sequence        = number-2              ; Cyclic sequence number
```

Fig. 8. PING OK Message that a Peer Send to Confirm it is Still Alive [35].

Bellow we explain the signification of the new part of the "PING OK" Message:

- Part 1: It is an Event Type (4 bytes), it is equal to %d7

F. Greeting Message over IPC

The following (Fig. 9) illustrates the Hello message in case of IPC Communication:

```
hello           = signature %d8 version sequence endpoint groups status name headers
signature       = %xAA %xA2             ; two octets
version         = number-1              ; Version number (2)
sequence        = number-2              ; Cyclic sequence number
uuids_services  = strings               ; list of nodes under the sender's responsibility with svc
groups          = strings               ; List of groups sender is in
status          = number-1              ; Sender groups status value
services        = strings               ; List of services offered by the sender
name            = string                ; Sender public name
headers         = dictionary            ; Sender header properties
```

Fig. 9. COPDAI Hello Message over IPC.

Bellow we explain the signification of the new part of the "Hello" Message over IPC:

- Part 1: It is an Event Type (4 bytes), it is equal to %d8.

G. Topology Heartbeating

Fig. 10 shows a typical example of the links between nodes in the COPDAI Middleware, nodes of the same hierarchical group communicate with each other and with their leader, leaders communicate with each other and with the Bridge Node, and finally Bridge Nodes communicate with each other.

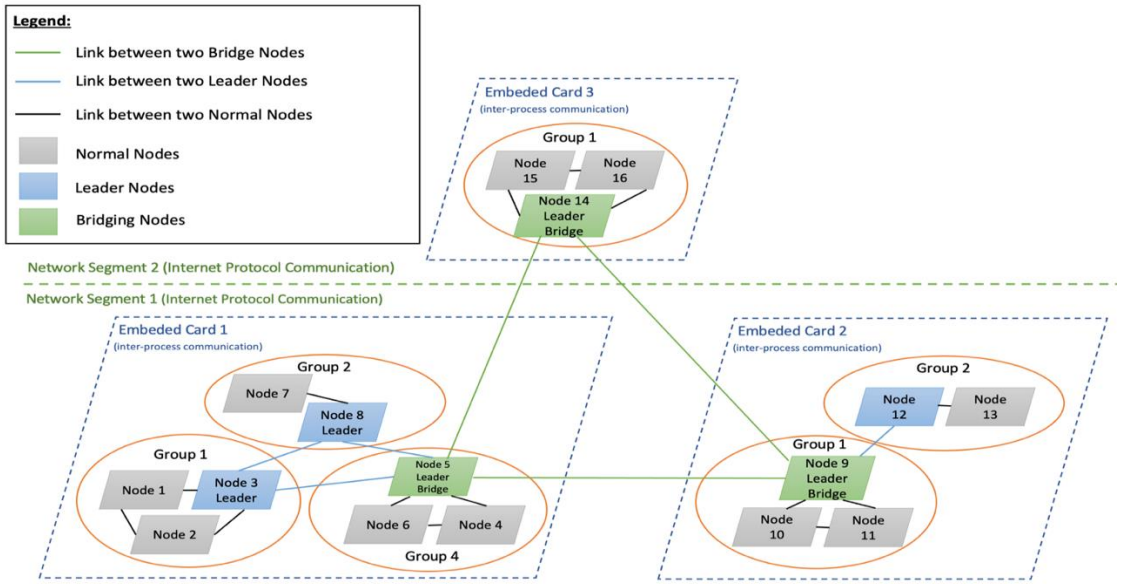

Fig. 10. Typical Communication Topology between COPDAI Nodes.

Each time the leader detects that there is a change in the nodes under its responsibility, e.g., a node in the group has disappeared, a new node has joined its group, it will notify the other leaders by sending the following message (Fig. 11):

```
topology        = signature %d9 version sequence topology status
signature       = %xAA %xA2             ; two octets
version         = number-1              ; Version number (2)
sequence        = number-2              ; Cyclic sequence number
uuids_services  = strings               ; list of nodes under the sender's responsibility with svc
groups          = strings               ; List of groups sender is in
status          = number-1              ; Sender groups status value
services        = strings               ; List of services offered by the sender
```

Fig. 11. COPDAI Topology Heartbeating Message.

Bellow we explain the signification of new part of the "Topology Heartbeating" Message:

- Part 1: It is an Event Type (4 bytes), it is equal to %d9

Bridge Node being itself a Leader, it is responsible for notifying the other Leaders in the same machine by any change in its group.

A Bridge Node is elected among the Leaders, so it is responsible for the propagation of the topology to others Bridge Nodes once a change has happened at the level of its group, or at the level of a group of another leader, the message (Fig. 11) is sent to others Bridge Nodes, with the difference that it concatenates all the nodes present on the machine with their respective services and not only the nodes that belong to its group.

In the opposite direction, once a Bridge Node receives a topology message from another one, it notifies the Leaders on its machine using the following message format (Fig. 12), in the same way the leaders propagate this message to each member of their group:

```
topology        = signature %d9 version sequence topology status
signature       = %xAA %xA2             ; two octets
version         = number-1              ; Version number (2)
sequence        = number-2              ; Cyclic sequence number
uuids_services  = strings               ; list of nodes under the sender's responsibility with svc
groups          = strings               ; List of groups sender is in
status          = number-1              ; Sender groups status value
services        = strings               ; List of services offered by the sender
```

Fig. 12. Remote Bridge Node Topology Heartbeating Message.

Bellow we explain the signification of new parts of the "Remote Bridge Node Topology Heartbeating" Message:

- Part 1: It is an Event Type (4 bytes), it is equal to %d10.
- Part 5: Remote Bridge Node UUID
- Part 6: List of UUIDs nodes under the responsibility of the Remote Bridge Node and for each UUID list of proposed services.
- Part 7: List of groups to which the Remote Bridge Node belongs.
- Part 9: List of services offered by the Remote Bridge Node.

H. Communication between Two Peers

One of the problems we have encountered in trying to have true Peer-to-Peer communication is that ZeroMQ socket is not symmetric, to overcome this problem, we have adopted the harmony pattern: For the outgoing messages, we are going to use a DEALER socket per peer so we can safely send messages.

For the ingoing messages, we choose the ROUTER socket, and so, the Harmony pattern comes down to these components (Fig. 13 and 14):

- One UDP socket where we listen to the broadcasted beacons (In case of Bridge Node).
- One ROUTER socket that we bind to an ephemeral port, and where we receive incoming messages from peers.
- One DEALER socket per peer that we connect to the peer's ROUTER socket.
- One ROUTER socket (named RT-ROUTER) that we bind to an ephemeral port, and where we receive incoming messages from peers which must be processed in real time (we suppose here that the Node is a type of RTCyclicNode and the listener is decorated properly to behave in real time (more details in our recent contribution [36]).
- One DEALER socket (named RT-DEALER) per peer that we connect to the peer's RT-ROUTER socket.
- Reading from our ROUTER/RT-ROUTER socket.
- Writing to the peer's DEALER/RT-DEALER socket.

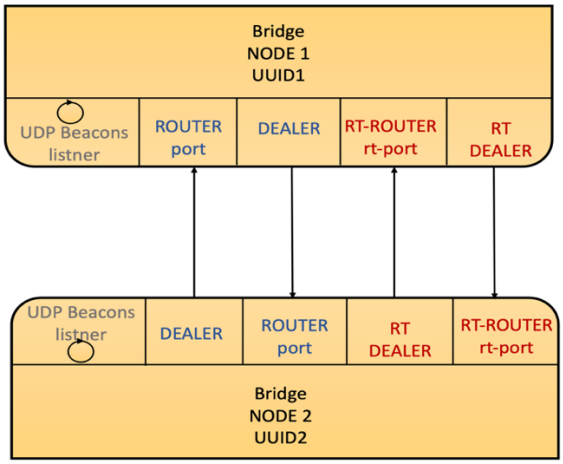

Fig. 13. Sockets used in each COPDAI Bridge Node (IP Communication).

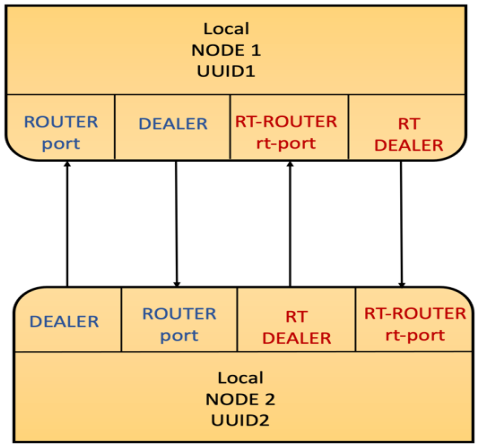

Fig. 14. Sockets used in each COPDAI Local Node (IPC Communication).

If the peer disappears and comes back with a different IP address and/or port, we have to disconnect our DEALER sockets and reconnect to the new ports.

In the case of IPC communication, a folder hierarchy was adopted as shown in (Fig. 15), a file is created in a folder named "dealer" which will be used as a medium for the DEALER socket, another file will be created in the folder "dealer/rt" for real time communication, the same tree structure is adopted for the ROUTER and RT-ROUTER sockets.

Fig. 15. IPC Sockets Folder Hierarchy.

The exchanged message between peers in the same hierarchical group is in this format (Fig. 16).

```
whisper             = signature %d2 version sequence content
version             = number-1              ; Version number (2)
service             = string                ; The service name to invoke
sequence            = number-2              ; Cyclic sequence number
content             = msg                   ; Wrapped message content
```

Fig. 16. Format of a Message Exchanged between two Nodes in Same Hierarchical Group.

Bellow we explain the signification of new parts of the Message exchanged between two nodes:

- Part 1: It is an Event Type (4 bytes); it is equal to %d2.
- Part 3: the service name to invoke.
- Part 6: the content message which is serialized using Protocol buffer [37] (it is the serialized object we are going pass to the service as a parameter).

I. COPDAI Node Topology Knowledge Management

Each node according to its position in the COPDAI hierarchy (Normal Node, Leader or Bridge), maintains some knowledge about the current topology:

- All Nodes maintains at least:
  - Peers UUIDs.
  - For each Peer UUID list of services offered by it.
  - For each Peer the list of groups to which it belongs.
  - For each Peer its Manager UUID (can be the UUID of a Leader or a Bridge Node).
  - For each Peer the time when it starts running (used in election).
  - List of Services.
  - For each Service list of Peers offering it.
  - For each Peer the outgoing message sequence (for assertion and Quality of Service (QoS)).
  - For each Peer the incoming message sequence (for assertion and QoS).
  - For each Peer group status sequence (for assertion and QoS).
  - Timestamp when this node start running (used in election).
  - The last time the node signaled its presence to the outside world (used in election).
- The Normal Node maintains also:
  - The UUID of the Leader Node that is responsible for it.
- The Leader Node maintains also:
  - Same Machine Leaders UUIDs.
  - The UUID of the Bridge Node that is responsible for it.
- Bridge Node maintains also:
  - Same Machine Leaders UUIDs.
  - Bridges Nodes UUIDs.
  - For each Bridge Node: The Endpoint, port, rt-port.

J. Message Routing

When a node wants to send a message to another node which does not belong to its hierarchical group, it constructs the message (Fig. 17) and sends it to the leader, if the leader finds that this node is managed by another leader on the same machine it sends the message to it, this last one transmits the message to the target node, if not, it transmits the message to the Bridge Node on the local Machine, this one will send the message to the Bridge Node which manages the target node, and thus the message is routed until it reaches its destination, an example is illustrated in Fig. 18:

```
route               = signature %d11 version routing content
version             = number-1              ; Version number (2)
target_uuid         = string                ; Target Node UUID
service             = string                ; The service name to invoke
sequence            = number-2              ; Cyclic sequence number
content             = msg                   ; Wrapped message content
```

Fig. 17. COPDAI Routing Message Format.

Bellow we explain the signification of new parts of the routing message:

- Part 1: It is an Event Type (4 bytes), it is equal to %d11.
- Part 3: The target peer.

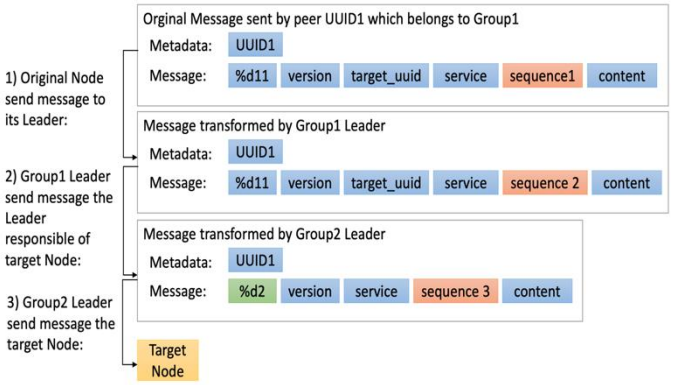

Fig. 18. Example of Routing Message between Nodes that doesn't belong to the same Hierarchical Group.

K. Group Messaging

For group messaging we want to be able to join and leave groups, discover the existing of nodes in other groups and send a message at once to several nodes belonging to the same group. This gives us some new protocol commands:

JOIN - we send this to all peers when we join a group (Fig. 19):

```
join        = signature %d4 version sequence group status
version     = number-1              ; Version number (2)
sequence    = number-2              ; Cyclic sequence number
group       = string                ; Name of group
status      = number-1              ; Sender groups status value
```

Fig. 19. Group Join Message Format [35].

- Bellow we explain the signification of new parts of the join message:
- Part 1: It is an Event Type (4 bytes), it is equal to %d4.
- Part 4: the group the node wants to join.

LEAVE - we send this to all peers when we leave a group (Fig. 20).

```
leave       = signature %d5 version sequence group status
version     = number-1              ; Version number (2)
sequence    = number-2              ; Cyclic sequence number
group       = string                ; Name of group
status      = number-1              ; Sender groups status value
```

Fig. 20. Leaving Group Message Format [35].

Bellow we explain the signification of new parts of the leave message:

- Part 1: It is an Event Type (4 bytes), it is equal to %d5,
- Part 4: the group the node wants to leave.

The following Fig. 21 illustrates the format of a message that will be sent to a group.

```
shout       = signature %d3 version sequence group content
version     = number-1              ; Version number (2)
sequence    = number-2              ; Cyclic sequence number
group       = string                ; Group to send to
service     = string                ; The service name to invoke
content     = msg                   ; Wrapped message content
```

Fig. 21. Multi-part Message Format for Group Messaging

Bellow we explain the signification of new parts of the multi-part message format:

- Part 1: It is an Event Type (4 bytes), if it is equal to %d3.
- Part 4: the group the node wants to send to it the message.
- Part 5: the service name to invoke.
- Part 6: Message content.

When a Leader receive a JOIN or LEAVE or a multi-part message it propagates it to other Leaders, the same for a Bridge Node if it receives a JOIN or LEAVE or multi-part message it propagates it to others Bridge Nodes.

Leaders and Bridge Nodes also are responsible for propagating this message to the Nodes they manage.

L. Election and Membership

1) When a node starts the first time, it sends a request to find a free group to all nodes in the same machine (Fig. 22).

2) If a leader receives a group search request, and if there is free space in its group, it reserves a space for the requester and sends an invitation (Fig. 23).

3) Once an invitation is received, the node sends a request to join the leader group (it must return the same invitation code received in the previous step) (Fig. 24).

4) Then the leader sends a confirmation with the name of the group that the node has just become one of its members (Fig. 25).

```
shout       = signature %d3 version sequence group content
version     = number-1              ; Version number (2)
sequence    = number-2              ; Cyclic sequence number
group       = string                ; Group to send to
service     = string                ; The service name to invoke
content     = msg                   ; Wrapped message content
```

Fig. 22. Search Free Group Message Format.

```
shout       = signature %d3 version sequence group content
version     = number-1              ; Version number (2)
sequence    = number-2              ; Cyclic sequence number
group       = string                ; Group to send to
service     = string                ; The service name to invoke
content     = msg                   ; Wrapped message content
```

Fig. 23. Leader Invitation Message Format.

```
shout          = signature %d3 version sequence group content
version        = number-1              ; Version number (2)
sequence       = number-2              ; Cyclic sequence number
group          = string                ; Group to send to
service        = string                ; The service name to invoke
content        = msg                   ; Wrapped message content
```

Fig. 24. Membership Request Format.

```
shout          = signature %d3 version sequence group content
version        = number-1              ; Version number (2)
sequence       = number-2              ; Cyclic sequence number
group          = string                ; Group to send to
service        = string                ; The service name to invoke
content        = msg                   ; Wrapped message content
```

Fig. 25. Membership Confirmation Message Format

5) Once this is done the node sends a "JOIN" message to notify everyone that it has just joined the group (Fig. 19), and after that it creates the necessary IPC sockets with other group's members.

Note: All these operations are time-stamped, if it takes times to receive a response, the operation is cancelled, and the process is resumed.

6) If the node doesn't receive any response from leaders, or if it has failed to become a member of a group after a configurable amount of time, the node creates a new group, joins it, and sends JOIN command to outside world, after that it becomes the leader of this new group it notifies peers with the following message (Fig. 26).

7) If Leaders receive a "LEADERSHIP" message, they send back a congratulation message, and specify which one of them is the Bridge Node (Fig. 27).

8) Once the Leader receives a congratulation message, it creates the necessary IPC sockets with the other Leader.

9) If after a while, the Leader doesn't receive any congratulation message, it considers itself as a Bridge Node, creates the necessary TCP sockets and starts listening on the UDP port to detect other Bridge Nodes over IP.

*10)*If a leader disappears, the rest of the nodes in the group start elections by exchanging between them the following message containing their start date (Fig. 28): the node that started first becomes the new leader and continues the process explained in step 6.

*11)*Each node saves/updates the last time it signalled its presence to the outside world, if it was a while since it notifies peers about its presence (a pre-parameterized value in COPDAI Middleware), and if it was a Leader, it concludes that he is no longer the leader and considers itself as a normal node and starts the process again from step 1.

*12)*If a Bridge Node disappears, the rest of the Leaders start elections by exchanging between them the message shown in (Fig. 28): The Leader that started first becomes the new Bridge Node and continues the process from step 9. The new Bridge Node is responsible of propagating the new topology to the outside world (Fig. 11).

```
shout          = signature %d3 version sequence group content
version        = number-1              ; Version number (2)
sequence       = number-2              ; Cyclic sequence number
group          = string                ; Group to send to
service        = string                ; The service name to invoke
content        = msg                   ; Wrapped message content
```

Fig. 26. Leadership Announcement Message.

```
shout          = signature %d3 version sequence group content
version        = number-1              ; Version number (2)
sequence       = number-2              ; Cyclic sequence number
group          = string                ; Group to send to
service        = string                ; The service name to invoke
content        = msg                   ; Wrapped message content
```

Fig. 27. Leaders Congratulation Message Format.

```
shout          = signature %d3 version sequence group content
version        = number-1              ; Version number (2)
sequence       = number-2              ; Cyclic sequence number
group          = string                ; Group to send to
service        = string                ; The service name to invoke
content        = msg                   ; Wrapped message content
```

Fig. 28. Election Message Format.

M. Content Sharing

We used the InterPlanetary File System (IPFS) [38] which is a peer-to-peer distributed file system that stores and retrieves files in a BitTorrent-like way.

So, to allow sharing of data captured by sensors (images, videos…) or artificial intelligence models between researchers / robots, we installed in each machine the ipfs daemon which connects it to the global distributed network by running the following commands:

$> ipfs init (1)

$> ipfs daemon (2)

IPFS requires 512MiB of memory and the installation takes only 12MB, if the machine doesn't have the necessary resources, we just ignore the IPFS installation.

The first time a node starts, it verifies if it has the ipfs capability by running the following command:

$> ipfs version (3)

If a node wants to add any file to the distributed file system, it just run:

$> ipfs add filename (4)

To allow nodes located on machines that do not have sufficient resources to share files, we run a dedicated COPDAI nodes (named IPFS Nodes) in servers that have enough resources, these nodes offer the "ipfs" service, and each node can send files to them using the "SEND MESSAGE" command (Fig. 16). The IPFS nodes persist the message content in file and after that, add it to the distributed file system (command 4).

After adding a file to ipfs, the command 4, returns a hash code, IPFS already offers the possibility to retrieve files via browser or command line interface (CLI) but it requires that we know already the hash code, to overcome this limitation, an event is fired associating each hash code with the UUID of the node that generated it and the file creation time, the event is sent to the distributed tracing system. In the future we plan to add an interface to brows the files by agents UUIDs / Names.

Researchers can share their content by just sharing the hash code.

N. Distributed Logging

In such a complex distributed system, tracing message between nodes is paramount, we have chosen Jeager [39] for distributed transaction and monitoring. The tracing is based on the OpenTracing Semantic Specification [40].

IV. Tests and Implementation

A first version of COPDAI Middleware is already developed in python [41], as well as in java [42], also an Android version of the COPDAI agent has been developed [43] to allow any robot to benefit from the existing sensors on a smartphone (Accelerometer, GPS, GYROSCOPE, Magnetometer…) and this allowed us to validate our communication architecture as well as to extend the capabilities of the robot used in our contribution [44] (Fig. 29) after attaching the smartphone to its body.

To challenge our middleware, we compared its performance with ROS2 [45] which is the upgrade of ROS1 by utilizing the Data Distribution Service, the main goal of ROS2 is to provide the real time capability, it is under heavy development, it supports communication over IP, ROS2 doesn't support ARM board even though most mobile robots use embedded cards based on ARM architecture like (Jetson TX2, Raspberry Pi, BeagleBone, Orange Pi, …), because they are energy efficient.

For each type of communication: COPDAI communication over IPC, COPDAI Real Time communication over IPC, Real Time communication over IP and communication using ROS2, we measured the latency that a message takes to pass from one node to another, we studied the following three scenarios: a node communicates only with one other node, a node communicates with 10 nodes and a node communicates with 100 nodes at the same time, for each scenario we sent 10k messages, to limit network noise, all nodes were deployed on the same machine (Asus Zephyrus ROG, CPU Pentium i7 2. 3GHz, RAM 16 GB) the RTOS used: Ubuntu 20.04 Patched to PREMPT_RT.

Table I illustrates the average latencies in each scenario, we notice that for the scenario of the real time communication using the COPDAI middleware on top of IPC the average latency did not change greatly by increasing from 10 nodes to 100 nodes, which shows a great stability of the system, on the other hand we notice that the average latency climbed in an exponential way in the case of ROS2, same for the case of the communication using COPDAI RT over IP or COPDAI over IPC the average latency is stable and robust to the scaling up.

Table II illustrates the maximum latencies obtained in each scenario, the highest latency was obtained when communicating between 100 nodes using ROS2 with more than 6 minutes of delay between sending and receiving the message, while we notice that the maximum latency in the case of using COPDAI RT over IP did not exceed 9 seconds and 7 seconds over IPC.

Table III illustrates the minimum latencies obtained in each scenario, communication between two nodes using COPDAI RT over IPC give the best result we also notice that in the case of 100 nodes for the same protocol we obtain a good result.

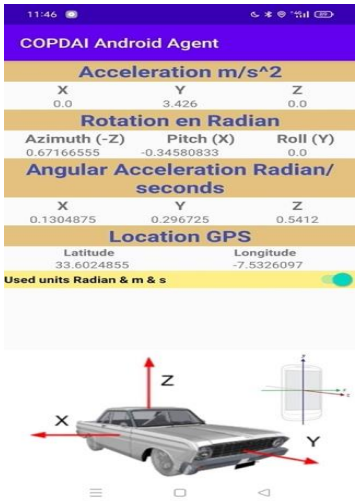

Fig. 29. COPDAI Android Agent which Enhance Robot's Capabilities by Sharing Sensors Data.

TABLE I. Average Latencies in (MS)

	One Node	10 Nodes	100 Nodes
COPDAI RT Over IPC	164.853	182.592	183.27
COPDAI RT Over IP	180.98	169.352	179.751
COPDAI Over IPC	200.878	201.494	198.819
ROS 2	306.636	415.334	1381.091

TABLE II. Maximum Latencies in (MS)

	One Node	10 Nodes	100 Nodes
COPDAI RT Over IPC	305.277	840.438	6865.242
COPDAI RT Over IP	389.717	1120.752	8817.04
COPDAI Over IPC	379.308	562.39	2322.554
ROS 2	51170.058	90730.255	412285.491

TABLE III. Minimum Latencies in (MS)

	One Node	10 Nodes	100 Nodes
COPDAI RT Over IPC	58.415	93.261	77.486
COPDAI RT Over IP	58.754	80.507	78.305
COPDAI Over IPC	103.863	109.328	111.111
ROS 2	181.098	163.54	273.133

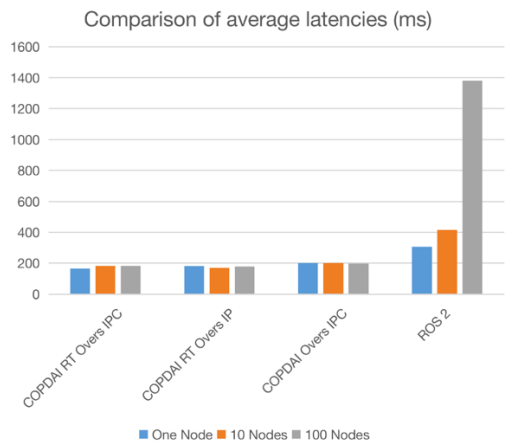

Fig. 30. Average Latency in Proportion with the Number of Nodes and the Communication Mechanism used.

As shown in Fig. 30, our middleware shows a very good performance, it scales efficiently, the real-time communication on IPC is the most optimized, which justifies our choice of such a mechanism for a communication within the same machine, and in general the real-time communication shows a great stability.

V. Conclusion

In this paper, a distributed, decentralized, real-time Peer-to-Peer protocol has been designed to allow robots and smart objects to act autonomously and improve their capabilities, COPDAI Middleware allows so, the Autonomous Unmanned Surface Vessels share their knowledge in extreme and hostile environments where links and components are subject to degradation. The designed protocol allows COPDAI nodes to build a mesh network and be aware of their environment. In addition, COPDAI solves the problem of the difficulty to have access to enough data and effectively train artificial intelligence models, by easily enabling the sharing of collected sensor data among the members of the scientific community. A first version of this Middleware has been developed in python, java and Android. We were also able to increase the perception capabilities of a mobile robot by attaching to its body an Android smartphone where COPDAI nodes are deployed, nodes collect mobile sensor data (Accelerometer, GPS, GYROSCOPE, Magnetometer...) and push them to the node deployed on the robot's embedded card. We compared the performance of COPDAI and the ROS2 Middleware. we found that COPDAI has a lower latency and better response time, in addition to a more stable communication when scaling the number of deployed nodes.

In our next work, we will look at the security of communication between nodes, and we will detail discovery and communication for nodes that are behind Firewalls or Routers.

References

[1] Vander Hook, J., Seto, W., Nguyen, V., Hasnain, Z., Gallagher, L., Halpin-Chan, T., Varahamurthy, V., & Angulo, M. (2019). Autonomous swarms of high speed maneuvering surface vessels for the central test evaluation improvement program. In Unmanned Systems Technology XXI (pp. 110210M). https://doi.org/10.1117/12.2518554.

[2] Esposito, J., Feemster, M., & Smith, E. (2008). Cooperative manipulation on the water using a swarm of autonomous tugboats. In 2008 IEEE International Conference on Robotics and Automation (pp. 1501–1506). https://doi.org/10.1109/ROBOT.2008.4543414.

[3] Langerwisch, M., Wittmann, T., Thamke, S., Remmersmann, T., Tiderko, A., & Wagner, B. (2013). Heterogeneous teams of unmanned ground and aerial robots for reconnaissance and surveillance-a field experiment. In 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR) (pp. 1–6). https://doi.org/10.1109/SSRR.2013.6719320.

[4] Patil, D., Upadhye, M., Kazi, F., & Singh, N. (2015). Multi robot communication and target tracking system with controller design and implementation of swarm robot using arduino. In 2015 International Conference on Industrial Instrumentation and Control (ICIC) (pp. 412–416). https://doi.org/10.1109/IIC.2015.7150777.

[5] Broberg, J., Hede, S., Mikkelsen, S., Pedersen, J., S\orensen, C., Madsen, P., & Borch, O. (2009). Collaboration Layer for Robots in Mobile Ad-hoc Networks. IFAC Proceedings Volumes, 42(22), 103–110. https://doi.org/10.3182/20091006-3-US-4006.00018.

[6] Iñigo-Blasco, P., Diaz-del-Rio, F., Romero-Ternero, M., Cagigas-Muñiz, D., & Vicente-Diaz, S. (2012). Robotics software frameworks for multi-agent robotic systems development. Robotics and Autonomous Systems, 60(6), 803–821. https://doi.org/10.1016/j.robot.2012.02.004.

[7] Chella, A., Cossentino, M., Gaglio, S., Sabatucci, L., & Seidita, V. (2010). Agent-oriented software patterns for rapid and affordable robot programming. Journal of systems and software, 83(4), 557–573. https://doi.org/10.1016/j.jss.2009.10.035.

[8] Kim, J.H., & Vadakkepat, P. (2000). Multi-agent systems: a survey from the robot-soccer perspective. Intelligent Automation & Soft Computing, 6(1), 3–17. https://doi.org/10.1080/10798587.2000.10768155.

[9] Kim, J.H., Shim, H.S., Kim, H.S., Jung, M.J., Choi, I.H., & Kim, J.O. (1997). A cooperative multi-agent system and its real time application to robot soccer. In Proceedings of International Conference on Robotics and Automation (pp. 638–643). https://doi.org/10.1109/ROBOT.1997.620108.

[10] KKalkhoff, W. (1995). Agent-Oriented Robot Task Transformation. In Proceedings of Tenth International Symposium on Intelligent Control (pp. 242–247). https://doi.org/10.1109/ISIC.1995.525066.

[11] Kuo, Y.h., & MacDonald, B. (2004). Designing a distributed real-time software framework for robotics. In Australasian Conference on Robotics and Automation (ACRA). Canberra.

[12] Schmidt, D. G., and Fred Kuhns. "An overview of the real-time CORBA specification." Computer 33.6 (2000): 56-63.

[13] Almeida, L., Santos, F., & Oliveira, L. (2016). Structuring communications for mobile cyber-physical systems. In Management of Cyber Physical Objects in the Future Internet of Things (pp. 51-76). Springer, Cham. https://doi.org/10.1007/978-3-319-26869-9_3.

[14] Muratore, L., Laurenzi, A., Hoffman, E., Rocchi, A., Caldwell, D., & Tsagarakis, N. (2017). Xbotcore: A real-time cross-robot software platform. In 2017 First IEEE International Conference on Robotic Computing (IRC) (pp. 77–80). https://doi.org/10.1109/IRC.2017.45.

[15] Munoz, M., Munera, E., Blanes, J., Simo, J., & Benet, G. (2013). Event driven middleware for distributed system control. XXXIV Jornadas de Automatica, 8.

[16] [16] Rabbah, M., Rabbah, N., Belhadaoui, H., & Rifi, M. (2016). Challenges facing middleware for mobile robots in smart environment. Int. J. Sci. Eng. Res, 7(11), 33–40.

[17] Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S., & Terziyan, V. (2008). Smart semantic middleware for the internet of things. In International Conference on Informatics in Control, Automation and Robotics (pp. 169–178).

[18] Choi, J., Cho, Y., Choi, J., & Choi, J. (2014). A layered middleware architecture for automated robot services. International Journal of Distributed Sensor Networks, 10(5), 201063. https://doi.org/10.1155/2014/201063.

[19] Fortino, G., Guerrieri, A., Lacopo, M., Lucia, M., & Russo, W. (2013). An agent-based middleware for cooperating smart objects. In International Conference on Practical Applications of Agents and Multi-

Agent Systems (pp. 387–398). https://doi.org/10.1007/978-3-642-38061-7_36.

[20] Savidis, A., & Stephanidis, C. (2003). Dynamic environment-adapted mobile interfaces: the Voyager Toolkit. Stephanidis, C.(Ed.), 4, 489–493.

[21] Dey, A., Abowd, G., & Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human–Computer Interaction, 16(2-4), 97–166. https://doi.org/10.1207/S15327051HCI16234_02.

[22] Brooks, R. (1997). The intelligent room project. In Proceedings Second International Conference on Cognitive Technology Humanizing the Information Age (pp. 271–278). https://doi.org/10.1109/CT.1997.617707.

[23] Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S., & Terziyan, V. (2008). Smart semantic middleware for the internet of things. In International Conference on Informatics in Control, Automation and Robotics (pp. 169–178).

[24] Choi, J., Cho, Y., Choi, J., & Choi, J. (2014). A layered middleware architecture for automated robot services. International Journal of Distributed Sensor Networks, 10(5), 201063. https://doi.org/10.1155/2014/201063.

[25] Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., De Souza, L., & Trifa, V. (2009). SOA-based integration of the internet of things in enterprise services. In 2009 IEEE international conference on web services (pp. 968–975). https://doi.org/10.1109/ICWS.2009.98.

[26] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A., & others (2009). ROS: an open-source Robot Operating System. In ICRA workshop on open source software (pp. 5).

[27] Suh, Y.H., Lee, K.W., & Cho, E.S. (2013). A device abstraction framework for the robotic mediator collaborating with smart environments. In 2013 IEEE 16th International Conference on Computational Science and Engineering (pp. 460–467). https://doi.org/10.1109/CSE.2013.75.

[28] Koubâa, A., Sriti, M. F., Bennaceur, H., Ammar, A., Javed, Y., Alajlan, M., ... & Shakshuki, E. (2015). Coros: A multi-agent software architecture for cooperative and autonomous service robots. In Cooperative Robots and Sensor Networks 2015 (pp. 3-30). Springer, Cham. https://doi.org/10.1007/978-3-319-18299-5_1.

[29] https://rfc.zeromq.org/spec/19/ (2021).

[30] Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561.

[31] ZeroMQ, https://zeromq.org/ (2021).

[32] Rizano, T., Abeni, L., & Palopoli, L. (2013). Experimental evaluation of the real-time performance of publish-subscribe middlewares.

[33] Lauener, J., & Sliwinski, W. (2017, October). How to design & implement a modern communication middleware based on ZeroMQ. In Proc of ICALEPCS (Vol. 17, pp. 45-51).

[34] Chang, H. J. (2015). A multi-agent message transfer architecture based on the messaging middleware zeromq. KIISE Transactions on Computing Practices, 21(4), 290-298. https://doi.org/10.5626/KTCP.2015.21.4.290.

[35] https://rfc.zeromq.org/spec/36/ (2021).

[36] Mahmoud Almostafa, R., Nabila, R., Hicham, B., & Mounir, R. (2018). Python in Real Time Application for Mobile Robot. Smart Application and Data Analysis for Smart Cities (SADASC'18). http://dx.doi.org/10.2139/ssrn.3179445.

[37] https://developers.google.com/protocol-buffers/ (2021).

[38] https://ipfs.io (2021).

[39] https://www.jaegertracing.io (2021).

[40] https://github.com/opentracing/specification/blob/master/specification.md (2021).

[41] https://github.com/mrabbah/copdaipythonagent (2021).

[42] https://github.com/mrabbah/jyre (2021).

[43] https://github.com/mrabbah/copdaiandroidagent (2021).

[44] Rabbah, M. A., Rabbah, N., Belhadaoui, H., & Rifi, M. (2017, October). Designing middleware over real time operating system for mobile robot. In First International Conference on Real Time Intelligent Systems (pp. 419-425). Springer, Cham. https://doi.org/10.1007/978-3-319-91337-7_37.

[45] Maruyama, Y., Kato, S., & Azumi, T. (2016, October). Exploring the performance of ROS2. In Proceedings of the 13th International Conference on Embedded Software (pp. 1-10). https://doi.org/10.1145/2968478.2968502.