Efficient Secure Communication for Distributed Multi-Agent Systems

Davide Costa[a], Daniel Garrido[b] and Daniel Castro Silva[c]
LIACC, Artificial Intelligence and Computer Science Laboratory, FEUP, Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal

Keywords: Multi-Agent, Middleware, Security, Distributed, Architecture.

Abstract: The use of multi-agent systems has been increasing and with it the need to improve communication performance and to make it secure. In a system with hundreds of agents, in which their reaction must be fast, it is essential to ensure low latency and high message throughput. These agents can work in a cooperative or competitive environment and, especially in the last, the absence of secure communications opens the possibility for malicious agents to intercept and/or change the content of the messages. This paper explores alternatives to the Message Transport Service as described in the Foundation for Intelligent Physical Agents architecture for multi-agent systems, namely using message-oriented middleware. It also introduces a new component to the typical multi-agent system architecture, the certification authority service. This component is responsible for creating certificates that platform agents can use to ensure their identity and communicate safely. This architecture also manages external agents distribution across several machines, similar to a federated environment, making the system more suitable for computationally demanding scenarios. The architecture was tested on an existing simulation platform, showing very good results.

1 INTRODUCTION

The increased use of multi-agent systems (MAS) for several applications, such as energy systems and industry 4.0, has led to the need for better performant communication middleware and the use of secure message exchange (Pires et al., 2018) (Xie and Liu, 2017). A system with a high number of agents in simultaneous communication and with the need to ensure (near) real-time communication requires a middleware that can ensure both low latency and high throughput (Calvaresi et al., 2017) (Leitão et al., 2013). Moreover, these agents can work in a competitive environment, in which secure communication stops malicious agents from intercepting and/or changing the contents of the messages being exchanged. This way, secure communications in MAS has been referenced in (Briones et al., 2016), (Tangod and Kulkarni, 2018), (Hedin and Moradian, 2015), (Subburaj and Urban, 2018) and other works as an important feature for current MAS.

The possibility of improving legacy applications by introducing MAS features has been referenced as a great advantage of MAS (Oprea, 2004). This can be done with the concept of external agents. External agents act like external applications not specifically developed for a MAS platform but that can connect to it and act as an agent. These agents have access to normal MAS functionalities like message exchange, yellow and white pages, allowing them to communicate and locate each other using a well-known standard promoted by FIPA[1] (Foundation for Intelligent Physical Agents).

In this paper we present an architecture that aims to introduce a new component in the FIPA architecture for MAS, the CA service. This service should create certificates that ensure agents' identities and allow them to communicate securely. We also study the integration of a communication middleware that can be used on a MAS. This middleware must provide the desired messaging performance (both in terms of latency and throughput) and implement security on message delivery ensuring confidentiality, integrity and authenticity.

To achieve that, some related work regarding multi-agent platforms, communication middleware and communication security will be explored in Section 2. A comparative analysis and some performance

[a] https://orcid.org/0000-0002-2190-5453
[b] https://orcid.org/0000-0002-7421-3119
[c] https://orcid.org/0000-0001-9293-0341

[1] More information available at http://fipa.org/

Costa, D., Garrido, D. and Silva, D.
Efficient Secure Communication for Distributed Multi-Agent Systems.
DOI: 10.5220/0010375605430552
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 1*, pages 543-552
ISBN: 978-989-758-484-8

tests will be conducted to show performance differences between message-oriented middleware and multi-agent platforms in Section 3, acting as state of the art regarding current MAS communication performance. The architectural solution, proposed in Section 4, should be generic, so it can be applied to any multi-agent system, endowing it with high performance and security on message delivery, while maintaining important features like compliance with a standard agent communication language and support for external agents. This is showcased in Section 5, with a case study. Section 6 presents some final conclusions on the developed solution and its abilities.

2 RELATED WORK

Some related works on multi-agent platforms, communication middleware and communication security are shown in the following sections.

2.1 Multi-Agent Platforms

Several studies on multi-agent platforms have been conducted, and detailed information can be found in (Kravari and Bassiliades, 2015) or (Leon et al., 2015). We focus on those platforms that are more referenced and adopted by the community, with current active development, support for secure communication (at least authenticity, confidentiality and integrity), following FIPA standards and with the possibility of running agents in different networked computers.

JIAC (Java-based Intelligent Agent Componentware): is an open-source framework and multi-agent architecture, developed in Java and distributed by DAI-Labor. JIAC uses, since version 5, Apache ActiveMQ as communication middleware. ActiveMQ (detailed below) is a message-oriented middleware that should provide JIAC with good performance (Lützenberger et al., 2015). JIAC has been used in several projects such as Energy Efficiency Controlling in the Automotive Industry (EnEffCo) (Küster et al., 2013), Intelligent Solutions for Protecting Interdependent Critical Infrastructures (ILIas) (Konnerth et al., 2012), Service Centric Home (SerCHo) (Hirsch et al., 2006) and others.

JADE (Java Agent DEvelopment Framework): is an open-source framework for multi-agent systems developed in Java and distributed by Telecom Italia (Telecom IT, 2017) (Kravari and Bassiliades, 2015). JADE uses a proprietary protocol for communication, the Internal Message Transport Protocol (IMTP). This protocol uses Java event-based communication within the same container and Remote Method Invocation (RMI) between containers (Soklabi et al., 2013) (Bellifemine et al., 2007). In JADE, a container represents a set of agents, and it is possible to have multiple containers distributed across the network. JADE has been widely referenced and used in several projects, such as Agreement Negotiation in Normative and Trust-enabled Environments (ANTE) (Lopes Cardoso et al., 2016) and a system that simulates dispatching and scheduling of transport and production tasks with automated guided vehicles in manufacturing systems (Braga et al., 2008).

SPADE (Smart Python Agent Development Environment): is an open-source platform for developing multi-agent systems in Python. Communication between two SPADE agents is based on instant messaging with Extensible Messaging and Presence Protocol (XMPP). This is a communication protocol for XML-based middleware (Gregori et al., 2006). SPADE is used in several projects such as a platform called SimFleet, a simulator for the use of car fleets in a more decentralized way (Carrascosa et al., 2019) and a recommendation system that analyzes user behaviour when visiting historical and cultural museum collections (Gelvez García et al., 2019).

2.2 Message Oriented Middleware Platforms

Several studies on message-oriented middleware platforms have been conducted, and detailed information can be found in (Celar et al., 2016) or (Yongguo et al., 2019). A survey on message oriented middleware was made in order to understand their characteristics, features, advantages and disadvantages. We focused on platforms that are more referenced in the literature, are free to use, have active development, and support for secure communication. Middleware platforms normally present an architecture with a broker. We can think of a broker as a post office: a message is sent to the broker and it then sends the message to the destination. However, some platforms use a brokerless architecture, using connections directly between sender and receiver, in a peer-to-peer (P2P) style.

ActiveMQ: is an open-source communication library developed by the Apache Software Foundation under the Apache License 2.0 (Apache Software Foundation, 2019d). There are currently two versions of this library: ActiveMQ 5 and ActiveMQ Artemis. We focus on Artemis as it will be the successor to ActiveMQ 5 as ActiveMQ 6 (Apache Software Foundation, 2019b). ActiveMQ Artemis uses a broker architecture in which the association to a queue or set of queues is made using the concept of address. The messages have an address that indicates where

they should be sent. Thus, sending a message to a given address may correspond to a send, by the broker, of the message to one or more queues associated with that address. There are two types of routing for a message: multicast, where each message is sent to all queues associated with that address; and anycast [2], where the message is only sent to one queue (Apache Software Foundation, 2019a). ActiveMQ has been used as communication middleware for several projects related to MAS such as in Decentralized Coordination Framework for Various Multi-Agent Systems (DeCoF) (Preisler et al., 2016) and also in the JIAC system, presented above.

Apache Kafka: is an open-source communication platform developed by LinkedIn for the Apache Software Foundation, under an Apache License 2.0 (Wang et al., 2015). Apache Kafka uses a broker architecture using the concept of topics. A topic serves as an address where records are published by producers. A record corresponds to a key, a value and a timestamp. A topic has several partitions and each partition consists of records that have been published there. A producer is responsible for choosing the partition where the record he produced will be published. The partition keeps the records in an orderly fashion, using a unique sequential identifier for each record (Apache Software Foundation, 2019c). A consumer can read a record by specifying the identifier of the record it wants to read from the partition. Apache Kafka has been used as communication middleware for several projects related to MAS such as a distributed data analysis framework for Industry 4.0 (Peres et al., 2016) and a distributed multi-agent application for continuous and integrated big data processing (Shashaj et al., 2019).

RabbitMQ: is an open-source communication platform developed by Pivotal Software under the Mozilla Public License (Pivotal, 2019). In RabbitMQ, sending messages from a producer to a consumer must always go through a broker, called Exchange. There are four types of exchanges (CloudAMQP, 2015): direct, fanout, topic and headers. RabbitMQ has been used as communication middleware for several projects related to MAS such as an ontology-based approach to scheduling jobs (Khegai et al., 2015) and a system to test multi-agent systems using a publish-subscribe architecture (Nascimento et al., 2017).

ZeroMQ: is an open-source communication library developed by iMatix Corporation under an LGPLv3 license (Lauener and Sliwinski, 2017). It has a brokerless architecture (Marcos, 2016), which brings, in theory, two advantages: i) a greater throughput, since the bottleneck for sending a larger number of messages is, in most cases, the lack of capacity of the broker; ii) lower latency, since the message is sent directly from producer to consumer, instead of sending it to the broker, who would process it and only then send it to the final recipient. ZeroMQ has been used as communication middleware in several projects related to MAS such as a system architecture for data and multimedia transmission for multi-UAV systems (Uk et al., 2018) and a system that allows to automate mission-level decision making for unmanned systems (Unmanned Systems Autonomy Services) (Li et al., 2018).

2.3 Communication Security

To mitigate security risks on communication, there are some requirements that a MAS must ensure (Borselius, 2002): integrity, ensuring there was no change in message content between send and arrival; confidentiality, ensuring the message content can only be perceived by authorized agents (i.e., those who have the key); authenticity, ensuring the message was in fact sent by whoever claims to be the author; and non-repudiation, ensuring that an agent cannot deny having sent a message it actually sent.

These requirements were covered in several MAS works such as (Poggi et al., 2001) and (Xu et al., 2010), with the use of public key cryptography (Mollin, 2002). Public key cryptography uses two keys: one public, known to everyone, and one private, that should be kept secret. The two keys are generated using some mathematical properties in a way that they are different but one can revert the effect of the other. Typical algorithms are elliptic curve and RSA, but elliptic curve manages to maintain the same level of security with much smaller keys when compared to RSA, making it more efficient and faster to compute (Sullivan, 2013).

As mentioned, the public key is known to everyone, but the association of a public key to an entity poses a problem: how to ensure a given public key actually belongs to that entity. The solution found to this vulnerability is the existence of trusted centralized authorities, who certify that a given public key really belongs to an entity by issuing a certificate[3]. This certificate, usually called digital certificate, corresponds to the entity's public key signed by the centralized authority with its private key, associated with an entity identifier. This certificate then allows everyone to verify, using the public key of the authority

[2]More information available at https://www.cloudflare.com/learning/cdn/glossary/anycast-network/

[3]More information available at https://www.venafi.com/education-center/pki/how-does-pki-work

(which is known to everyone), that a given public key really belongs to an entity (Henmi, 2006).

Another type of cryptography is private key cryptography, in which the same key is used to both encrypt and decrypt the content of a message. This type of encryption is much faster than public key encryption (according to (Haque et al., 2018), 1000 times faster) and also more secure (Mollin, 2002). However, it does not consider key exchange.

Transport Layer Security (TLS[4]) is a commonly used protocol for secure communication, using a hybrid approach of public and private encryption. For a given session between two parties, this protocol uses public key encryption for authentication and exchange of a secure symmetric key; then, for the rest of the session, that symmetric key is used for private key encryption, as it is more efficient and secure.

3 EXPERIMENTAL PERFORMANCE EVALUATION

To better understand if a more efficient and secure solution could be found with the use of a message-oriented middleware, we performed some experiments to compare the performance of the studied message-oriented middleware and MAS. These experiments were done both without and with the use of security (TLS), also providing us with insights on the performance penalty it entails.

The experiments were performed for three typical agent communication scenarios (Paletta, 2012): one-to-one; broadcast; and many-to-one. The tests were carried out with 10, 50 and 100 pairs of agents communicating simultaneously in the one-to-one communication scenario; 10 producers with 10, 50 and 100 consumers and 50 producers with 10, 50 consumers in the broadcast communication; and 10, 50 and 100 producers sending to only one consumer in many-to-one communication.

The content of each message is its own timestamp. A group with 500 messages was sent by each producer, iteratively with an interval of 1ms between each message. The use of this interval makes the test case closer to the real one and has a great impact on the performance values of the middleware.

The tests on communication middleware were performed using ActiveMQ Artemis 2.10.1, RabbitMQ 3.8.2 and Apache Kafka 2.4.0 server versions, and Apache.NMS.ActiveMQ 1.7.2, RabbitMQ.Client 5.1.2, Confluent.Kafka 1.3.0, NetMQ 3.3.3.4 and clrzmq4 4.1.0.31 C# .Net clients. The use of C# clients is related to compatibility with the case study presented in section 5. The tests on MAS were performed using JIAC 5.2.4, JADE 4.5.0 with JADE-S 3.10 and SPADE 3.1.4. All tests were performed on a computer with an Intel 4720HQ@2.6Ghz CPU, with 16GB of RAM @1600MHz, 500GB SSD and Windows 10 operating system.

In one-to-one communication, results on Table 1 show ZeroMQ as the best in terms of latency (results in the format Latency / Throughput). In terms of throughput, ZeroMQ when using security suffers a very significant impact and RabbitMQ takes the lead. Regarding ZeroMQ, it is also necessary to refer the use of two C# clients, NetMQ in the insecure version and clrzmq in the secure one. Despite being the recommended client, and having recently received support for the use of elliptic curve encryption, NetMQ does not yet support the use of certificates for client authentication.

Table 1: One-to-One Performance Tests. Values represent latency (ms) and throughput (msg/s).

Secure	Platform	10 Pairs	50 Pairs	100 Pairs
No	ActiveMQ	11 / 1830	23 / 2004	26 / 3358
	A. Kafka	574 / 9763	747 / 6512	1475 / 7525
	ZeroMQ	0 / 4771	1 / 20342	4 / 23321
	RabbitMQ	1 / 4980	11 / 12742	27 / 12791
	JADE	3 / 2393	14 / 7319	104 / 12376
	JIAC	1317 / 763	9223 / 1512	17437 / 2125
	SPADE	468 / 3771	1302 / 7542	2706 / 14321
Yes	ActiveMQ	18 / 1905	181 / 1968	429 / 1981
	A. Kafka	565 / 4244	618 / 1901	1166 / 2871
	ZeroMQ	3 / 2657	13 / 2360	23 / 3282
	RabbitMQ	1 / 4789	19 / 8951	33 / 11529
	JADE	33 / 1679	125 / 2811	1016 / 5792
	JIAC	1583 / 744	14224 / 1301	18567 / 1871
	SPADE	553 / 2649	1978 / 4360	3978 / 9282

Some other platforms also presented good results: RabbitMQ has good latencies and good throughput; ActiveMQ has good latencies but a low throughput when compared to other middleware; JADE is the only MAS with good overall results (but still far from most middleware in most cases). Apache Kafka, SPADE and JIAC present worse results, a situation that repeats itself in other communication scenarios. Apache Kafka presents unexpectedly bad results, much higher than all other middleware. The overall cost of using security was an increase of 21.38% in latency and a decrease of 51.3% on throughput.

In broadcast communication, results on Table 2 show RabbitMQ as the best. ZeroMQ's latency has increased a lot with the need of using a broker for broadcast – its documentation recommends the use of

[4]More information available at https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/

a broker to solve the "Dynamic Discovery Problem"[5]. In terms of throughput, ZeroMQ is now superior in both insecure and secure communication, by a large margin. Regarding other platforms, we refer to the overall good results of JADE but mainly for the scenario of 50 publishers and 50 subscribers, in which results are very close to those of RabbitMQ. The overall cost of using security was an increase of 13.33% in latency and a decrease of 52.7% on throughput.

Table 2: Broadcast Latency Performance Tests. Values represent latency (ms) and throughput (msg/s).

Secure	Platform	10 Pubs 10 Subs	10 Pubs 50 Subs	10 Pubs 100 Subs	50 Pubs 50 Subs	50 Pubs 50 Subs
No	ActiveMQ	5216/31	7095/64	7259/70	10305/192	9583/896
	A. Kafka	50000/604	18154/3211	13488/3977	11640/3901	6894/9557
	ZeroMQ	181159/3	454546/237	372024/425	146456/904	441889/1473
	RabbitMQ	40683/2	63516/11	75896/32	39494/41	76108/279
	JADE	13466/14	16159/32	12634/381	34699/207	34882/271
	JIAC	870/1267	1004/6096	1269/18656	981/9156	1866/18943
	SPADE	2791/1065	2802/3370	3398/6685	3058/3768	3262/9529
Yes	ActiveMQ	3659/300	2927/82	2458/273	2504/130	2552/921
	A. Kafka	4163/1115	12303/2898	13488/4265	10493/4294	3699/11042
	ZeroMQ	47755/4	166778/340	190986/831	76617/1429	167515/3658
	RabbitMQ	34435/5	50648/17	45741/103	48049/33	48005/293
	JADE	6154/123	10940/126	11121/1603	21988/350	18207/375
	JIAC	884/1502	954/7433	1155/19674	1060/9355	1484/21365
	SPADE	2760/1115	2880/3898	3135/7265	2992/4294	3269/11042

In many-to-one communication, results on Table 3 show JADE as the overall best in terms of latency without the use of security (results in the format Latency / Throughput), followed by RabbitMQ. When using security, RabbitMQ has better latencies. ZeroMQ and JADE, with very similar performance between them, are the ones that follow RabbitMQ but with significantly higher values. The other middleware and MAS have much higher results. In terms of throughput, ZeroMQ is superior in all scenarios. The overall cost of using security was an increase of 21.39% in latency and a decrease of 91.41% on throughput.

Table 3: Many-to-One Performance Tests. Values represent latency (ms) and throughput (msg/s).

Secure	Platform	10 Pubs	50 Pubs	100 Pubs
No	ActiveMQ	11 / 808	28 / 2365	381 / 3897
	A. Kafka	617 / 64103	768 / 27503	1628 / 12034
	ZeroMQ	13 / 454546	58 / 446429	186 / 310559
	RabbitMQ	1 / 5005	13 / 18382	28 / 25013
	JADE	4 / 2589	7 / 10549	14 / 13763
	JIAC	1574 / 1072	11747 / 1917	19475 / 2555
	SPADE	133 / 1771	741 / 10342	2079 / 11321
Yes	ActiveMQ	9 / 931	67 / 1466	3787 / 1764
	A. Kafka	448 / 4382	782 / 5569	1319 / 5105
	ZeroMQ	16 / 4916	65 / 14501	201 / 16955
	RabbitMQ	1 / 4921	17 / 10233	47 / 16361
	JADE	11 / 2152	57 / 4428	176 / 7409
	JIAC	1818 / 856	17361 / 1659	21096 / 2176
	SPADE	140 / 2109	830 / 6192	1907 / 8437

[5]More information available at http://zguide.zeromq.org/page:all#header-39

Table 4 shows an overall comparison of all tested platforms. The first two lines show the number of times a platform has been the best in all scenarios for insecure and secure versions, respectively. Results are presented as Latency / Throughput. The third and fourth lines show the number of times a platform reached the top 3 in a scenario, allowing for a better understanding of which platforms perform the best overall. The last two lines show the percentage difference of the secure version when compared to the insecure one (the fifth line shows latency and the sixth throughput), providing a clearer picture of the performance impact in each platform.

These results show RabbitMQ as the most consistent in all scenarios, and also to have better performance than all tested MAS platforms. Considering these results, the quality of its documentation and the fact that the second best choice (ZeroMQ) would imply using two C# clients, RabbitMQ was chosen.

Table 4: Overall Comparison (Ins. for Insecure and Sec. for Secure).

Metric	ActiveMQ	A. Kafka	ZeroMQ	RabbitMQ	JADE	JIAC	SPADE
1st Ins.	0 / 0	0 / 1	3 / 10	5 / 0	3 / 0	0 / 0	0 / 0
1st Sec.	0 / 0	0 / 0	3 / 7	8 / 4	0 / 0	0 / 0	0 / 0
Top 3 Ins.	7 / 0	0 / 6	5 / 11	11 / 11	10 / 3	0 / 0	0 / 2
Top 3 Sec.	7 / 0	0 / 5	7 / 11	11 / 11	8 / 2	0 / 0	0 / 4
Latency	603%	-7%	235%	39%	632%	18%	19%
Throughput	-41%	-68%	-80%	-25%	-47%	-11%	-22%

4 PROPOSED ARCHITECTURE

We propose a multi-agent system in which RabbitMQ acts as the Message Transport Service (MTS[6]) of the FIPA architecture for MAS, being responsible for all message transfers between agents. In order to secure yellow and white pages operations, RabbitMQ will also transport messages regarding these operations, but the operation itself continues to be provided by the original MAS. This works as a wrapper around operations' messages, with confidentiality and integrity assurance for all operations and also sender identity verification when registering, modifying and de-registering a service. In this architecture, all agents are connected as publishers, subscribers or both, allowing them to send messages to each other. When starting an agent, the system will create a queue for the agent, which other agents can use to send messages to it. A channel is also created that can be used by the agent to send messages. Two main communication scenarios are presented, for individual messaging and broadcast.

[6]More information available at http://www.fipa.org/specs/fipa00067/SC00067F.html#_Toc26669817

The **individual scenario** is the most simple case of communication, messaging from one agent to another. In this scenario, we use a direct exchange in which a message arriving at the exchange is sent to the queues whose routing key is the same as the routing key of the message (Marcos, 2016). Each agent has a unique routing key (its AID, or Agent Identifier), so any agent just needs to know the recipient's AID (which can be learned using the yellow or white pages) to communicate with it.

The **broadcast scenario** uses broadcast to send messages to all agents interested in a topic. This scenario uses a fanout exchange in which a message reaching an exchange is sent to all queues linked to it (Marcos, 2016). An agent only needs to send a message to the correct exchange (different exchange per topic) using a routing pattern for the topic. In our case we want a message sent to an exchange to be sent to all subscribers, so we need to have n queues bound to the exchange, one per subscriber agent.

In order to fulfill our objectives we needed to add to this base FIPA architecture support for secure communication. RabbitMQ supports the use of security between sender and broker and between broker and receiver. However, it is necessary to provide each agent with a certificate in which the RabbitMQ broker trusts because it was signed by a certificate authority (CA) it knows and trusts. The certificate also needs to contain agent-specific information such as AID. To allow this, our architecture introduces a new component, a CA service, that will sign a certificate for each valid agent.

Multi-agent systems were designed with a distributed architecture in mind, allowing to distribute agents across several machines. Typically, multi-agent platforms allow to do that with internal agents by default. However, when using external agents this is not assured. In order to allow distribution for external agents, while considering the secure certification architecture already presented, we also developed a distribution architecture. We have called Remote Node to a node/computer where agents can be placed to run. This solution is similar to the distribution of the load on a MAS platform using a federation with multiple MAS platform instances (Wang and Zhang, 2012). This mechanism works with multiple Remote Nodes to allow high demanding scenarios, but also works with the simple case of just one Remote Node with all agents. This Remote Node can even be on the same machine as the Master node with the system working as a package, like a big single component. In this distributed architecture, an agent's certificate needs to contain the IP address of the machine where the agent is running (the machine from which the agent will connect to the RabbitMQ broker). If an agent migrates to another Remote Node it will have its certificate regenerated for the new IP address. We considered that a multi-agent platform with support for distribution needs to have at least two roles: a Master node, the central entity of the system responsible for launching the agents; and the agents themselves.

To allow all agents to have valid certificates we had to create an integrated architecture that allows an agent to ask for certification. It is also necessary that the CA only issues certificates to trusted agents. The process adopted to achieve this is depicted in Fig. 1.

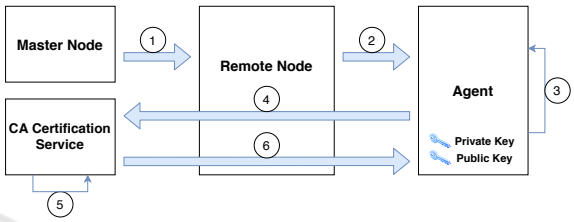

Figure 1: Agent Certification Architecture.

In **step 1**, the Master node distributes the agents by all available Remote Nodes in order to achieve load balance. The communication between the Master node and each Remote node can be carried out securely through the use of certificates with RabbitMQ. The Master node as well as the Remote nodes previously had digital certificates generated on their behalf and signed by the system's CA. In terms of scalability, as there is only one Master node and the Remote nodes growth is controlled by it, this is an easily practiced precondition.

In **step 2**, each Remote node will start the execution of the received agents.

In **step 3**, each agent, at the time of its creation, is responsible for the generation of a pair of keys, used later to ensure its identity. This pair consists in a public key and a private key.

In **step 4**, each agent sends its public key to the system CA to create a valid certificate through a new component that works as a service. For this communication to happen safely, the created agents will, only for this request and since an agent is a Remote node's child, send its public key to be signed on behalf of the Remote node. The goal is to ensure that only agents that have been created by system Remote nodes (and that way considered trusted) have their certificates signed and further access to the platform.

In **step 5**, the system's CA generates a certificate for each received request, verifying they came from a trusted Remote node.

In **step 6**, each certificate is sent to the agent, who

can use it in future communications with other agents. These agents of the platform trust these certificates because they trust in the identity that issued it (the system's CA). This sending operation is carried out securely through RabbitMQ using certificates from the system CA and the agent's Remote node (as the signed certificate is public, and anyone can see it, this is only a security measure to prevent integrity issues).

After these steps, it is now possible to guarantee that messages are exchanged with confidentiality and integrity. However, in terms of authenticity we only ensure that who sends the message is an agent with a certificate trusted by the RabbitMQ broker. As RabbitMQ uses a broker, the final message receiver doesn't have access to the certificate used by the original sender and so it can't validate its identity. However, this can be solved using a RabbitMQ property, user-id[7], that can be sent together with the message. This property can be set by the sender to indicate its identity but a message is only successfully sent if the logged in user has in fact logged in with a username equal to user-id. Thus, the use of this property implies that every agent registers itself in the broker.

RabbitMQ offers a plugin that helps in user management, allowing to register a user just using a username and no password. To login, a user has to use a certificate in which the broker trusts and that has a Distinguished Name (DN) equal to the username used. The DN[8] is an identifier that uniquely identifies an entity in a certificate. This identifier is built using other fields of the certificate like CommonName (CN), UserId (UID) and others.

The registration imposes two challenges: having certificates to all the platform agents with DN equal to the username registered; and secure communication with the RabbitMQ management plugin for registration. The certificates architecture presented before already solves the first, as long as the generated certificates have the correct DN. A good DN format could be "UID=<AgentId>,CN=<AgentIpAddress>" and so this needs to be the username with which the agent is registered. The use of the agent's IP address as the CN is necessary because RabbitMQ checks if the IP from which it is receiving the request matches that value. The UID field of a certificate can be used for storing the agent id (AID), allowing the receiver to check it. The registration of an agent in RabbitMQ can be done securely using the available HTTPS API, by the Master node before launching the agent.

RabbitMQ allows setting permissions regarding access to queues and exchanges, which are divided in three: configure (related to declaring operations), read and write (Pivotal, 2020). When registering a user we used these permissions to only allow the necessary operations to that agent, improving security by using authorisation. Only an agent with ID equal to X should be able to declare a queue in some standard format, like "Agent-X-MessageQueue", to read messages. However, all agents should be able to write to it, to communicate with others.

Since the beginning we knew that using security would have a performance penalty that the previous performance tests shown. So, we have planned to make possible the use of different security assurance levels to balance between security and performance as needed. The idea is that the Master node can choose between different types of security on a given execution. The levels of security available are: authentication + SSL, just SSL and no security.

5 CASE STUDY

To test the proposed solution we used a simulation platform (SimPlatform) previously developed by this research group for simulation of cooperative missions using heterogeneous autonomous vehicles (Silva, 2011). This platform uses a FIPA-compliant multi-agent system, AgentService[9], with no security in message exchange. We made a comparison between the performance of SimPlatform with AgentService and now with RabbitMQ (both insecure and secure versions). The tests were divided in three scenarios as before: one-to-one and many-to-one at Table 5 and broadcast at Table 6.

Table 5: One-to-One and Many-to-One Before and After Comparison. Values represent latency (ms) and throughput (msg/s).

Test	Platform	10 pairs	50 pairs	100 pairs
One-To-One	**AgentService**	2216 / 466	14416 / 422	31922 / 388
	RabbitMQ Ins.	2 / 4916	15 / 6342	36 / 4884
	RabbitMQ Sec.	5 / 4344	26 / 5421	46 / 3732
Many-To-One	**AgentService**	7428 / 338	115677 / 117	685647 / 39
	RabbitMQ Ins.	2 / 4995	13 / 11611	35 / 8258
	RabbitMQ Sec.	4 / 4985	36 / 10365	61 / 7365

The tests show a tremendous improvement both in latency and throughput, even when using security, which was not used before. AgentService's values are much worse than all other MAS tested before (JADE, JIAC and SPADE). However, it needs to be considered that AgentService was used with external agents

[7]More information available at https://www.rabbitmq.com/validated-user-id.html

[8]More information available at https://tools.ietf.org/html/rfc4514

[9]Archived image available at https://web.archive.org/web/20170703015137/http://www.agentservice.it/

Table 6: Broadcast Before and After Comparison. Latency in ms and Throughput in msg/s.

Test	Platform	10 Pubs 10 Subs	10 Pubs 50 Subs	10 Pubs 100 Subs	50 Pubs 10 Subs	50 Pubs 50 Subs
Latency	AgentService	24992	128926	308930	428928	1652217
	RabbitMQ Ins.	12	30	96	43	274
	RabbitMQ Sec.	19	63	289	59	383
Throughput	AgentService	965	944	744	339	436
	RabbitMQ Ins.	27762	33607	20055	24512	25849
	RabbitMQ Sec.	14885	21404	14550	23558	19879

and the other MAS with normal in-platform agents. The values presented by RabbitMQ, when compared to the tests made with RabbitMQ alone, suffered some performance penalty related to the use of larger messages and the cost of message serialization. However, RabbitMQ continues to present the best overall results when compared to all other middleware and MAS. With these changes, and the new middleware, SimPlatform is now capable of providing real-time simulations with many more agents and in a secure way.

6 CONCLUSIONS

This paper presents the integration of a message-oriented middleware in a multi-agent system to improve communication efficiency, and also to provide support for secure message exchange with the introduction of CA certification service on FIPA standard architecture for MAS. The middleware choice was done after performance tests on a selected group of MAS and communication middleware platforms. JADE, the studied MAS with best results, presented on average 7 times higher latencies and half the throughput when compared to RabbitMQ. To the best of our knowledge, this is the first solution that uses RabbitMQ to replace a multi-agent system middleware. We believe this solution could be implemented in other multi-agent systems with very good results, as was the case of our case study, SimPlatform. We also describe how a simulation platform with external agents, i.e., external applications with MAS functionalities, can have its performance improved using agents distribution over several machines and how it can be done securely.

Future improvements could be made to this architecture mostly regarding availability and security. Regarding fault tolerance, this architecture presents three potential points of failure: RabbitMQ broker failure; Master node failure; CA certification service failure. RabbitMQ broker failure can be solved with the use of a broker cluster, which also improves performance, as described in (Jack Vanlightly, 2018). Both Master node failure and CA certification service failure can be solved with the use of multiple instances/nodes of each one working with a mechanism called "Role Exchange" between a set of master and slave nodes, as described in (Gankevich et al., 2016). This mechanism allows one slave node to take over the master's role/position when it fails. This way it becomes possible to have better performance and availability.

Regarding security, the developed system allows receiving agents to verify the AID and IP address of the agent who sent the message. Although safe, this system does not include the agent's decision process on whether or not to trust the sender of a given message. One approach to this would be a mechanism referenced in the literature as distributed trust, in which agent A adjusts the level of trust in another agent B based on the veracity of the information it has been receiving from B (Dorri et al., 2018).

REFERENCES

Apache Software Foundation (2019a). Core. Available at https://activemq.apache.org/components/artemis/documentation/latest/core.html (accessed Nov. 2019).

Apache Software Foundation (2019b). How does ActiveMQ compare to Artemis. Available at https://activemq.apache.org/how-does-activemq-compare-to-artemis (accessed Nov. 2019).

Apache Software Foundation (2019c). Introduction. Available at https://kafka.apache.org/intro (accessed Nov. 2019).

Apache Software Foundation (2019d). Legal Notice. Available at https://activemq.apache.org/components/artemis/documentation/1.5.0/notice.html (accessed Nov. 2019).

Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

Borselius, N. (2002). Security in multi-agent systems. In *Proceedings of the 2002 International Conference on Security and Management (SAM'02), June, Las Vegas, Nevada, 2002*, pages 31–36.

Braga, R., Rossetti, R., Reis, L. P., and Oliveira, E. (2008). Applying multi-agent systems to simulate dynamic control in flexible manufacturing scenarios. In *Proceedings of 19th European Meeting on Cybernetics and Systems Research (EMCSR 2008). Mar. 25-28, Vienna, Austria, 2008*, pages 488–493.

Briones, A. G., Chamoso, P., and Barriuso, A. (2016). Review of the Main Security Problems with Multi-Agent Systems used in E-commerce Applications. *Advances in Distributed Computing and Artificial Intelligence Journal*, 5(3):55–61.

Calvaresi, D., Marinoni, M., Sturm, A., Schumacher, M., and Buttazzo, G. (2017). The Challenge of Real-Time Multi-Agent Systems for Enabling IoT and CPS. In *Proceedings of the 2017 International Conference on*

Web Intelligence (WI'17), Aug. 23-26, Leipzig, Germany, 2017, pages 356—-364.

Carrascosa, C., Terrasa, A., Palanca, J., and Julián, V. (2019). SimFleet: A New Transport Fleet Simulator Based on MAS. In De La Prieta, F., González-Briones, A., Pawleski, P., Calvaresi, D., Del Val, E., Lopes, F., Julian, V., Osaba, E., and Sánchez-Iborra, R., editors, *Proceedings of International Workshops of Practical Applications of Survivable Agents and Multi-Agent Systems (PAAMS 2019), June 26-28, Ávila, Spain, 2019*, pages 257–264.

Celar, S., Mudnic, E., and Seremet, Z. (2016). State-of-the-art of messaging for distributed computing systems. In *Proceedings of 27th International Symposium on Intelligent Manufacturing and Automation (DAAAM'16), 26-29 Oct., Mostar, Bosnia and Herzegovina, 2016*, volume 27, pages 298–307.

CloudAMQP (2015). Part 4: RabbitMQ Exchanges, routing keys and bindings. Available at https://www.cloudamqp.com/blog/2015-09-03-part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html (accessed Nov. 2019).

Dorri, A., Kanhere, S. S., and Jurdak, R. (2018). Multi-Agent Systems: A Survey. *IEEE Access*, 6:28573–28593. DOI: 10.1109/ACCESS.2018.2831228.

Gankevich, I., Tipikin, Y., Korkhov, V., Gaiduchok, V., Degtyarev, A., and Bogdanov, A. (2016). Factory: Master Node High-Availability for Big Data Applications and Beyond. In Gervasi, O., Murgante, B., Misra, S., Rocha, A. M. A., Torre, C. M., Taniar, D., Apduhan, B. O., Stankova, E., and Wang, S., editors, *Proceedings of 16th International Conference on Computational Science and Its Applications (ICCSA 2016), July 4–7, Beijing, China, 2016*, pages 379–389.

Gelvez García, N. Y., Ballén Duarte, A. D., and Espitia Cuchango, H. E. (2019). Multi-Agent System Used for Recommendation of Historical and Cultural Memories. *Tecciencia*, 14:43–52.

Gregori, M. E., Cámara, J. P., and Bada, G. A. (2006). A Jabber-Based Multi-Agent System Platform. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06), May 8-12, Hakodate, Japan, 2006*, page 1282–1284. DOI: 10.1145/1160633.1160866.

Haque, M. E., Zobaed, S., Islam, M. U., and Areef, F. M. (2018). Performance Analysis of Cryptographic Algorithms for Selecting Better Utilization on Resource Constraint Devices. In *Proceedings of 2018 21st International Conference of Computer and Information Technology (ICCIT'18), 21-23 Dec., Dhaka, Bangladesh, 2018*.

Hedin, Y. and Moradian, E. (2015). Security in Multi-Agent Systems. In *Proceedings of 19th Annual Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2015), Sept. 7-9, Singapore, 2015*, volume 60, pages 1604–1612.

Henmi, A. (2006). *Firewall Policies and VPN Configurations*, chapter Chapter 5 - Defining a VPN, pages 211–265. Syngress, Burlington.

Hirsch, B., Konnerth, T., Heler, A., and Albayrak, S. (2006). A Serviceware Framework for Designing Ambient Services. In *Proceedings of the First International Conference on Ambient Intelligence Developments (AmID'06), Sept. 20-22, Sophia Antipolis, France, 2006*, pages 124–136. Springer, Paris.

Jack Vanlightly (2018). RabbitMQ vs Kafka Part 5 - Fault Tolerance and High Availability with RabbitMQ Clustering. Available at https://jack-vanlightly.com/blog/2018/8/31/rabbitmq-vs-kafka-part-5-fault-tolerance-and-high-availability-with-rabbitmq (accessed May 2020).

Khegai, M., Zubok, D., and Maiatin, A. (2015). Ontology-Based Approach to Scheduling of Jobs Processed by Applications Running in Virtual Environments. In *Proceedings of 6th International Conference on Knowledge Engineering and Semantic Web (KESW 2015), Sept. 30 - Oct. 2, Moscow, Russia, 2015*, pages 273–282.

Konnerth, T., Chinnow, J., Kaiser, S., Grunewald, D., Bsufka, K., and Albayrak, S. (2012). Integration of Simulations and MAS for Smart Grid Management Systems. In *Proceedings of the 3rd International Workshop on Agent Technologies for Energy Systems (ATES 2012), June 5, Valencia, Spain, 2012*, pages 51–58.

Kravari, K. and Bassiliades, N. (2015). A Survey of Agent Platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):18. DOI: 10.18564/jasss.2661.

Küster, T., Lützenberger, M., and Freund, D. (2013). Distributed Evolutionary Optimisation for Electricity Price Responsive Manufacturing using Multi-Agent System Technology. *International Journal On Advances in Intelligent Systems*, 6(1&2):27–40.

Lauener, J. and Sliwinski, W. (2017). How to design & implement a modern communication middleware based on ZeroMQ. In *Proceedings of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17), Oct. 8-13, Barcelona, Spain, 2017*.

Leitão, P., Inden, U., and Ruckemann, C.-P. (2013). Parallelising Multi-agent Systems for High Performance Computing. In *Proceedings of Third International Conference on Advanced Communications and Computation (INFOCOMP'13), Nov. 17-22, Lisbon, Portugal, 2013*, pages 1–6.

Leon, F., Paprzycki, M., and Ganzha, M. (2015). A Review of Agent Platforms. Technical report, CT COST Action IC1404, Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS).

Li, Y., Frasure, I., Ikusan, A. A., Zhang, J., and Dai, R. (2018). Vulnerability Assessment for Unmanned Systems Autonomy Services Architecture. In Au, M. H., Yiu, S. M., Li, J., Luo, X., Wang, C., Castiglione, A., and Kluczniak, K., editors, *Proceedings of 12th International Conference on Network and System Security (NSS 2018), Aug. 27-29, Hong Kong, China, 2018*, pages 266–276.

Lopes Cardoso, H., Urbano, J., Rocha, A. P., Castro, A. J. M., and Oliveira, E. (2016). *ANTE: A Framework Integrating Negotiation, Norms and Trust*, pages 27–45.

Lützenberger, M., Konnerth, T., and Küster, T. (2015). *Industrial Agents*, chapter 21 - Programming of Multia-

gent Applications with JIAC, pages 381–398. Morgan Kaufmann, Boston.

Marcos, P. B. (2016). Resources Management Monitorization Platform on a Message Oriented Middleware System. Master's thesis, University of Porto.

Mollin, R. A. (2002). *RSA and Public-Key Cryptography*. Discrete Mathematics and Its Applications. CRC Press, Inc., USA.

Nascimento, N. M., Viana, C. J., von Staa, A., and Lucena, C. (2017). A Publish-Subscribe based Architecture for Testing Multiagent Systems. In *Proceeding of 29th International Conference on Software Engineering and Knowledge Engineering (SEKE'17), July 5-7, Pittsburgh, PA, USA, 2017*, pages 521–526.

Oprea, M. (2004). Applications of Multi-Agent Systems. In Reis, R., editor, *Proceedings of International Federation for Information Processing 18th World Computer Congress (WCC2004), August 22-27, Toulouse, France, 2004*, pages 239–270.

Paletta, M. (2012). Self-Organizing Multi-Agent Systems by means of Scout Movement. *Recent Patents on Computer Science*, 5(3):197–210.

Peres, R. S., Rocha, A. D., Coelho, A., and Barata Oliveira, J. (2016). A Highly Flexible, Distributed Data Analysis Framework for Industry 4.0 Manufacturing Systems. In *Proceedings of 6th International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing (SOHOMA 2016), Oct. 6-7, Lisbon, Portugal, 2016*, volume 694, pages 373–381.

Pires, F., Barbosa, J., and Leitão, P. (2018). Quo Vadis Industry 4.0: An Overview Based on Scientific Publications Analytics. In *Proceedings of 2018 IEEE 27th International Symposium on Industrial Electronics (ISIE 2018), June 13-15, Cairns, QLD, Australia, 2018*, pages 663–668.

Pivotal (2019). Mozilla Public License. Available at https://www.rabbitmq.com/mpl.html (accessed Jan. 2020).

Pivotal (2020). Authorisation: How Permissions Work. Available at https://www.rabbitmq.com/access-control.html#authorisation (accessed May 2020).

Poggi, A., Rimassa, G., and Tomaiuolo, M. (2001). Multi-User and Security Support for Multi-Agent Systems. In *Proceedings of 2nd Workshop From Objects to Agents (WOA 2001), Sep. 4-5, Modena, Italy, 2001*, pages 13–18.

Preisler, T., Dethlefs, T., and Renz, W. (2016). DeCoF: A Decentralized Coordination Framework for Various Multi-Agent Systems. In Klusch, M., Unland, R., Shehory, O., Pokahr, A., and Ahrndt, S., editors, *Proceedings of 4th German Conference on Multiagent System Technologies (MATES'16), Sept. 27–30, Klagenfurt, Austria, 2016*, volume 9872, pages 73–88.

Shashaj, A., Mastrorilli, F., Morrelli, M., Pansini, G., Iannucci, E., and Polito, M. (2019). A Distributed Multi-Agent System (MAS) Application For continuous and Integrated Big Data Processing. In Chatzigiannakis, I., De Ruyter, B., and Mavrommati, I., editors, *Proceedings of 15th European Conference on Ambient Intelligence (AmI 2019), Nov. 13–15, Rome, Italy, 2019*, pages 350–356.

Silva, D. C. (2011). *Cooperative Multi-Robot Missions: Development of a Platform and a Specification Language*. PhD thesis, University of Porto, Porto, Portugal.

Soklabi, A., Bahaj, M., and Cherti, I. (2013). JIAC Systems and JADE Agents Communication. *International Journal of Engineering and Technology*, 5(2):1976–1984.

Subburaj, V. H. and Urban, J. E. (2018). Specifying Security Requirements in Multi-agent Systems Using the Descartes-Agent Specification Language and AUML. In *Proceedings of 15th Conference on Advanced Information Technologies for Management (AITM 2018) and 13th Conference on Information Systems Management (ISM 2018), Sept. 9-12, Poznan, Poland, 2018*, pages 93–111.

Sullivan, N. (2013). A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography. Available at https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography (accessed Jan. 2020).

Tangod, K. and Kulkarni, G. (2018). Secure Communication through MultiAgent System-Based Diabetes Diagnosing and Classification. *Journal of Intelligent Systems*, 29(1):703–718.

Telecom IT (2017). JADE. Available at https://jade.tilab.com/ (accessed Nov. 2019).

Uk, B., Konam, D., Passot, C., Erdelj, M., and Natalizio, E. (2018). Implementing a System Architecture for Data and Multimedia Transmission in a Multi-UAV System. In Chowdhury, K. R., Di Felice, M., Matta, I., and Sheng, B., editors, *Proceedings of 16th International Conference on Wired/Wireless Internet Communication (WWIC 2018), June 18–20, Boston, MA, USA, 2018*, pages 246–257.

Wang, G., Koshy, J., Subramanian, S., Paramasivam, K., Zadeh, M., Narkhede, N., Rao, J., Kreps, J., and Stein, J. (2015). Building a Replicated Logging System with Apache Kafka. *Proc. VLDB Endow.*, 8(12):1654–1655. DOI: 10.14778/2824032.2824063.

Wang, X. and Zhang, L. (2012). Multi-Agent Systems Simulation Base on HLA Framework. In Lee, G., editor, *Proceedings of International Conference on Automation and Robotics (ICAR 2011), Dec. 1-2, Dubai, United Arab Emirates, 2011*, pages 339–346.

Xie, J. and Liu, C.-C. (2017). Multi-agent systems and their applications. *Journal of International Council on Electrical Engineering*, 7(1):188–197.

Xu, X., Xiong, J., and Cheng, C. (2010). The model and the security mechanism of the information retrieval system based on mobile multi-agent. In *Proceedings of 2010 IEEE 12th International Conference on Communication Technology (ICCT'10), Nov. 11-14, Nanjing, China, 2010*, pages 25–28.

Yongguo, J., Qiang, L., Changshuai, Q., Jian, S., and Qianqian, L. (2019). Message-oriented Middleware: A Review. In *Proceedings of 5th International Conference on Big Data Computing and Communications (BIGCOM'19), 9-11 Aug., QingDao, China, 2019*, pages 88–97. DOI: 10.1109/BIGCOM.2019.00023.