Evaluation and comparison of middleware DDS technologies

João Aires, Miguel Barros, Pedro Costa
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
{up201404269, up201404278, up201402793}@fe.up.pt

***Abstract*—In this project, a DCPS (Data-Centric Publisher-Subscriber) service was implemented using a middleware DDS (Data Distribution Service). This service was implemented using two different middlewares: OpenDDS and RTI.**

Complementary to this, performance measurements were carried out for both technologies. Additionally, a comprehensive comparison, encompassing both performance and other non-functional characteristics, was performed.

I. Introduction

This section provides a theoretical background used during the implementation of this project: middleware technologies, the publisher-subscriber cooperation model, and an overview of DDS.

A. Middleware

Middleware consists of a layer of software with the purpose of simplifying application development. This is performed by enabling communication and management of data in distributed applications, thus abstracting its implementation from the purpose of the application, and with the goal of being platform-agnostic in mind. As a consequence, platform details are hidden, and new services are added to the system.

Because of this additional software layer, non-functional properties (such as timing, performance and dependability) are affected. The impact on these properties will be significantly influenced by the quality of the middleware being used.

B. Cooperation model: Publisher-Subscriber

A distributed system is composed by two main components: the communication system (of which its protocol structures can be achieved by middleware) and its computing nodes. In order to exchange information in the system, the computing nodes can use a range of different cooperation models. One example of this is the Publisher-Subscriber model.

In this model, communication is performed by means of topics, in which:

- One or more *producers* process information and trigger transactions (i.e. disseminates information).
- One or more *subscribers* consume the information sent by the producers.

C. DDS and middleware implementations

The behaviour described above can be replicated by means of a DDS. This consists of an open specification, proposed by OMG [1]. It implements a DCPS by means of a global distributed database constituted by topics. An overview of the system is shown in Figure 1.

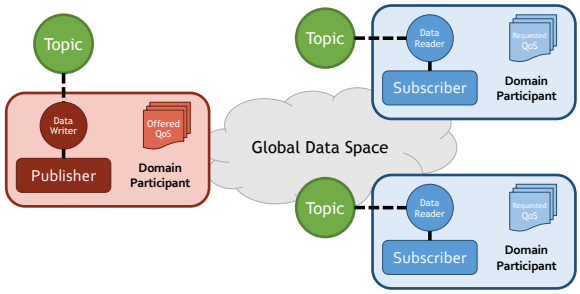

Fig. 1. Overview of a DDS system

The main components of the DDS are:

- *GDS (Global Data Space)*: distributed data space accessible by all applications.
- *Topic*: description of data to be published and subscribed, of which its information can be produced or consumed.
- *Publisher*: responsible for dissemination of data.
- *Subscriber*: responsible for reception of data.
- *DataWriter*: entity that declares intention to publish to topic.
- *DataReader*: entity that declares intention to receive from topic.

Some implementations of DDS are available, namely:

- *OpenDDS* [2]: complete, open-source implementation of the protocol
- *RTI* [3]: commercial implementation of the protocol

Both implementations were used for creating a DCPS system in this project.

II. Implementation

This section provides some information on the implementations performed. This involved creating a DCPS layer using two different DDS implementations: OpenDDS and RTI.

A. Procedure

In this system, general nodes are created, which are defined by an `ID`. Each node contains both publisher and subscriber functionalities. Specifically, for `ID = 0`, the node is responsible for periodically ping-ing all other nodes, which act as subscribers, and fetching latency measurements to a `.dat` file. By doing this, a single clock is used, which avoids measurement errors. Because the order of magnitude of the latency is comparable with the magnitude of clock differences, even small changes could translate to significant measurement errors.

Pings are performed once every second during 30 seconds. Data communication between nodes is performed using UDP multicast.

Timestamp calculation is performed as follows:

- Node `ID = 0` sends its timestamp to the topic, which forwards the message to the subscribers.
- The subscribers send back the message with the original timestamp.
- The publisher receives back the messages, which contain each subscriber's ID, and locally computes the new timestamp and determines the delay.

A high-level explanation of this algorithm follows in Listing 1. The final code was written in C++.

```
initialises subcriber
initialises publisher

if node_id equals 0: /* node is responsible for
    computing RTD */
{
  sends ping message with timestamp
  waits for all nodes
  computes round-trip-delay[node] = time_now -
      time_sent
}
else:
{
  waits for message
  fetches timestamp from received message
  sends message with ACK and received timestamp
}
```

Listing 1. Pseudo-code for the operations on each node

The system is scalable, so it can have any number N of instantiated nodes. It is also possible for the nodes to be instantiated in different systems, as will be discussed in section III, and using different DDS implementations.

An example of the implemented system is shown in Figure 2. The structure of the messages used is shown in Figure 3.

B. Computational setup

The following setup was used for testing purposes:

- *Processor*: 7th gen. dual-core Intel i5 processor
- *Memory*: 8GB RAM
- *Hard disk*: 128GB SSD
- *Operating System*: Mac OS X (Darwin)

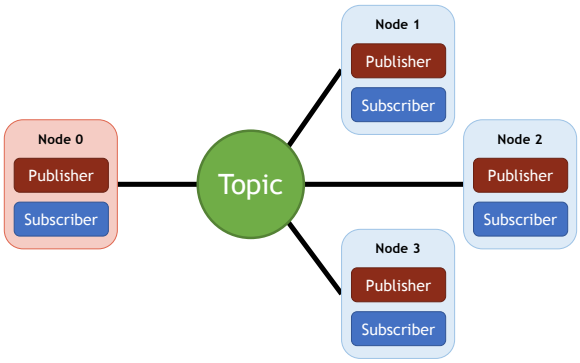

Fig. 2. Example of the DDS system for N = 3 subscriber nodes

Node ID	Timestamp	Data
(int)	(int)	(variable)

Fig. 3. Structure of messages sent between nodes

III. Results and Discussion

In this section, the test results of both OpenDDS and RTI DDS implementations are presented.

A. Single system

Below follow the results for N nodes in a single system.

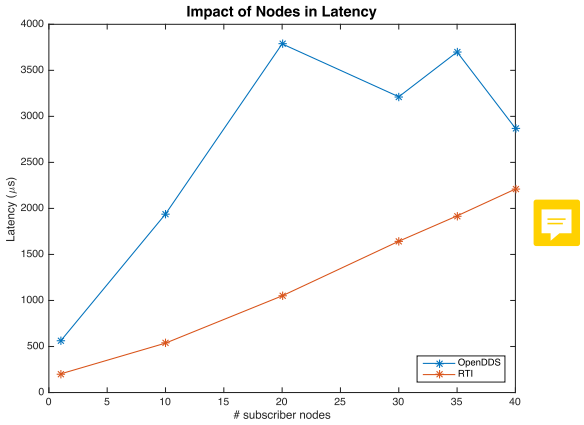

Fig. 4.

Upon analysing this data, it is concluded that RTI provides better performance in comparison with OpenDDS. RTI is not only faster, but it better mimics a deterministic behaviour. This is specially important when taking into consideration the IoT, mission-critical scenarios that DDS systems are to undertake.

The reasons for this are as follows:

- *Mean node latency*, in Figure 4, increases linearly with the increase of nodes for RTI. OpenDDS mean node latency increases at a higher rate, and in a more unpredictable fashion.
- *Mean latency values for a large number of nodes*, in Figure 5, offer a more stable behaviour for RTI. On the

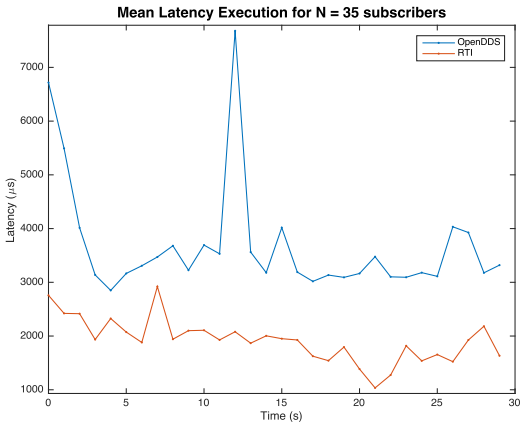

Fig. 5.

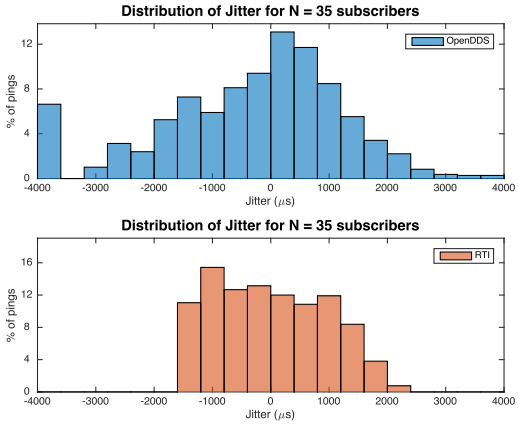

Fig. 6.

one hand, its baseline value is lower in comparison with OpenDDS. On the other hand, the values are also more stable with time. Notably, for t = 12s, a spike is registered for OpenDDS. This likely happened due to a concurrent system execution, which took away resources needed for quickly executing the OpenDDS instance.

- *Distribution of jitter*, in Figure 6, is noticeably more compact for RTI. Maximum jitter variation for RTI corresponds to around 2ms, whereas for OpenDDS values above 12ms were registered. Besides this, a significant percentage of values, of 7%, were registered at around -4000µs, for OpenDDS. This is a result of some nodes registering a near-zero latency value, and of a high latency baseline.

During testing, it was observed that significant delays in the system occur for N $>$ 40 nodes, for both OpenDDS and RTI. This constitutes a bottleneck for implementing the DDS standard.

B. Multiple systems

Additionally, a comparison was performed for two systems, with N = 1 subscriber node. This comparison was performed by connecting both systems to a common switch. Additionally, `iPerf` was used in order to stimulate data traffic in the network.

A *latency execution* was performed for RTI, and the results obtained are shown in Figure 7.

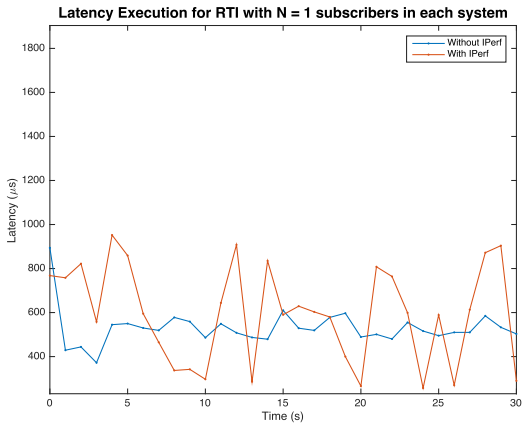

Fig. 7.

The results allow to conclude that adding traffic to the system slows the algorithm, as the baseline for the first scenario is lower. Noticeably, however, the values with *iPerf* measurements vary significantly. This is likely due to the low rate used for data transfer.

On the other hand, when comparing with Figure 4, it is also concluded that the system is around 2x slower when nodes are located in different machines. Namely, in this scenario the data packets need to traverse all OSI layers, since data is routed to outside the machines (and to the switch).

C. Non-functional middleware requirements

To complement the aforementioned performance analysis, a more comprehensive non-functional analysis was performed. The results of this analysis, along with the set requirements, are shown in Table I.

It was noted that significant effort is needed in order to use both implementations. Part of this effort comes from the usage of structures, used to guarantee a platform-agnostic software. Besides this, more learning material exists for RTI, and its API is better documented than for OpenDDS. However, the high cost of RTI, here used using an academic license, may render it unsuitable for project implementation purposes.

For both implementations, resource utilisation is low. This makes it possible to run the software in more resource-constrained systems, such as a Raspberry Pi.

Requirements	OpenDDS	RTI
Simplification of application development	Steep learning curve	Steep learning curve
API documentation	Limited	Comprehensive
Installation process	Command-based in CLI	Automated GUI
Setup process	Hardware abstraction	Hardware dependent
Resource utilisation	Low	Low
Price	Free	Subscription-dependent
Technical support	None	Subscription-dependent
Integration with low-power devices	Yes	Yes

TABLE I
NON-FUNCTIONAL REQUIREMENTS COMPLIANCE FOR OPENDDS AND RTI

IV. FUTURE IMPROVEMENTS

Upon performing this project, some additional improvements and opportunities for future works were identified.

- Porting both DDS implementations to Raspberry Pi is possible. This may allow for a project oriented towards an IoT scenario.
- Testing with other DDS implementations.
- Exploring more QoS policies. DDS offers a wide range of policies, of which a small portion was used during this project.
- Impact of different transport protocols.

V. GROUP MANAGEMENT

The work on this project was divided between each member as follows:

- João Aires: RTI implementation, non-functional requirements and guidelines (33%)
- Miguel Barros: OpenDDS implementation, non-functional requirements and guidelines (33%)
- Pedro Costa: performance metrics and plotting, presentation and report (33%)

REFERENCES

[1] "What is dds?" [Online; accessed 16 Jan. 2018]. [Online]. Available: https://www.omgwiki.org/dds/what-is-dds-3/

[2] "Opendds," [Online; accessed 17 Jan. 2018]. [Online]. Available: http://opendds.org

[3] "Connectivity software framework for the industrial iot — rti," [Online; accessed 17 Jan. 2018]. [Online]. Available: https://www.rti.com