NAPA: An algorithm to auto-tune unicast reliable communications over DDS

Juan J. Martin-Carrascosa, Jose M- Lopez-Vega, Javier Povedano-Molina,
Juan J. Ramos-Muñoz, Juan M. Lopez-Soler
Signal Theory, Telematics, and Communications Department
ETSI Informática y Telecomunicación
University of Granada
SPAIN

***Abstract*: This paper proposes NAPA (Non-supervised Adaptative Publication Algorithm) a framework for auto-tuning unicast reliable communications over DDS. We provide the NAPA design rationale, and some implementation details. After the experimental conducted evaluation, we demonstrate how using the subscriber's feedback, as NAPA does, the publisher can vary its sending rate in order to improve the overall performance in terms of end-to-end latency and throughput in DDS applications.**

Keywords: dds; throughput; latency; tuning; middleware;

I. INTRODUCTION

This paper focuses on the publish-subscribe interaction model. In the publish-subscribe model, the publish-subscribe service decouples the information producers (publishers) from information consumers (subscribers). Publishers update a shared information space and subscribers indicate what data are they interested in. It's the publish-subscribe service who delivers the desired information from publishers to subscribers.

Publish-subscribe systems can be classified according to the mechanism they use to choose what the events of interest for the different entities are. They can be topic-based (events classified according to a label), content-based (events classified according to their content) or type-based (event classified according to their structure).

Data Distribution Service (DDS) [1] is a middleware specification adopted by the Object Management Group (OMG) [4] aimed to standardize data-centric publish/subscribe communications in distributed scenarios. DDS was specified in 2004 and that specification includes the advances developed by different companies until the moment. This specification defines an Application Programming Interface (API) designed for data distribution in real-time using publish-subscribe model.

Over the last few years, DDS has been deployed in many different contexts ranging from mission critical systems to healthcare systems or financial systems. All these scenarios have in common that different remote entities exchange data from different sources in a well controlled environment and with certain guarantees like reliability and determinism.

Publish/subscribe communication systems are able to de-couple in time and space the publishers and subscribers. The existence of projects like PSIRP [5] and PURSUIT[6] evidence the potential benefits of this model. These projects are aimed to define a publish-subscribe based Internet that can be considered a plausible alternative (even complementary) to the current design.

DDS defines a virtual data space –typically implemented as a distributed cache memory- where applications can share information just by writing or reading data identified by a name (topic), a type and optionally a key. Using this virtual data space reduces the difficulty implementing communications between system nodes, what eases the designing of distributed systems.

DDS is based on a data-centric model (where data is not opaque to the middleware) and provides a rich set of Quality of Service (QoS) that define the requirements of communications on each different scenario.

DDS specification defines two different interface levels and a layer for interoperability.

The **Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol (DDS-RTPS)** interoperability layer. This layer allows different implementations of DDS to interoperate. It defines the discovery protocol, data representation format and message format. This layer was specified by the OMG in order to provide interoperability between the different implementations of DDS.

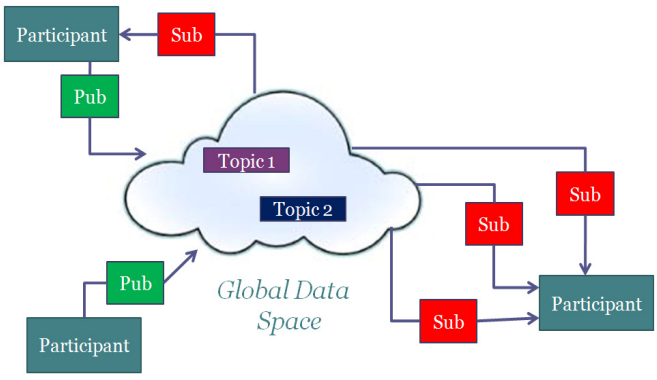

Fig 1. An overview of DDS entities.

The DCPS layer defines the concept of Data-space, Topic (group of data of same type), Publishers (data producers) and Subscribers (data consumers) (see Fig. 1). Also, this layer introduces the support of QoS Policies, group of settings that determines the behavior of the middleware and releases the application developers of certain tasks such as communication reliability.

DDS scenarios may vary considerably. In this regard, system administrators usually need to tune the middleware and nodes forming the full system in order to get better performance in terms of throughput and latency.

Although manual tuning may improve the performance achieved, tuned parameters' sensibility and scenario's variability (network variations, number of nodes and CPU load in nodes) reduce the effectiveness of the tuning. This issue could even discourage system administrators from performing any adjustment at all.

In this paper we propose an algorithm for dynamically tuning reliable communications over DDS. In particular, this algorithm adjusts data publishing rate according to subscriber's feedback (Acknowledge messages).

Regarding the performance and evaluation, to the author's knowledge there aren't any benchmarks universally accepted for evaluating real-time scenarios. However, some initiatives are gaining popularity for the case of real-time system benchmarking. Two of these famous benchmarks are SPECjms2007 [7] and STAC-M2 [8]. Nevertheless, these benchmarks are for MOM (Message Oriented Software), not for data-centric middleware.

In addition to these, the authors have already published other works related to DDS and its performance and evaluation. Most related publications are "A content-aware bridging service for publish/subscribe environments" [9] and "Performance evaluation of Publish/Subscribe middleware technologies for ATM (air traffic management) systems" [10].

The remainder of this paper is organized as follows. In Section 2, we elaborate about the reasons that motivate researching on this topic. Section 3 describes the proposed algorithm design. In Section 4 we explain the Implementation details. In Section 5 we show the results obtained when evaluating our algorithm. Finally, in Section 6 the main conclusions of this paper are shown.

II. MOTIVATION

By definition, in reliable communications involved nodes do their best to provide data with no errors, typically by resending lost samples. The additional generated traffic reduces the available bandwidth for the main communication and increases the latency both in the resent samples and the following ones. This is mainly due to two effects of sample resending: first, the increase of buffer waiting time; and secondly, the blocking of the publisher while it waits to receive the delayed data acknowledgments.

Losses are not the only reason of transmission blocking in reliable communications. In the DDS middleware, publishers have a size-limited send window that blocks communication when it is full. Since DDS does not removes samples from the window until the subscriber has acknowledged them, publishers get blocked when they send data faster than subscribers can receive and process it.

For this reason, sending data as fast as possible is not the optimal behavior, due to it can cause getting the publisher blocked. It means that those samples --being ready to be sent-- experiment an increased latency due to the incurred buffer waiting time. Note that latency is not only affected by the time spent in the link (transmission and propagation), but also the incurred time between different communication layers.

These are the reasons why there is a known need of tuning communication between publishers and subscribers: if publishers send data at the maximum rate that subscribers allow, resends' traffic will be reduced, as well as time spent in buffers.

The effect of this additional traffic is shown in Fig. 2. This Figure depicts throughput and one-way latency in function of the elapsed time between consecutive sent data, in a 1 to 1 reliable communications. In this case we use 100kB messages. As it can be seen, waiting enough time between consecutive samples may result in a better performance than sending data as fast as possible. However, if the publisher waits too much time, throughput will get worse although latency is reduced considerably.

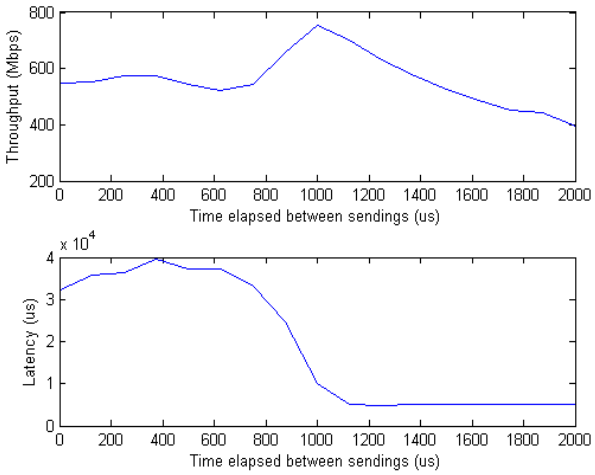

Fig 2. Performance *versus* inter-packet elapsed time.

In Figure 2 there is a range of elapsed time values that provides the best trade-off, that is, better throughput and latency. However, this range of value is dynamic and depends on sample data size, network status, CPU usage, etc.

This paper studies these DDS performance issues and particularly proposes an auto-tuning algorithm for making the system to work in the optimal operation point.

III. ALGORITHM DESIGN

The purpose of using reliable communications is to ensure that the subscriber receives all the samples (in order, with no errors either losses). Analyzing results shown in the previous section, we can enunciate the following statements:

1. As the inter-packet elapsed time is reduced, subscriber can potentially receive them faster than it can process. In this case, it starts rejecting new incoming samples and it will ask for retransmissions. As a consequence, the throughput gets lower and latency will accordingly increase.
2. For large inter-packet elapsed times, the publisher will be sending at a lower rate than the subscriber can receive. Throughput is decreased due that low rate, but latency is smaller because the system is more relaxed and there is no need to wait so much time in buffers.
3. In the correct operation point, publisher sends samples at the same rate that the subscriber can process. This makes throughput be better because retransmissions are not needed and latency gets minimized because there is no congestion in the communication path.

Now that the effect of varying the time elapsed between packets is characterized, we analyze what are the requirements that must accomplish the tuning algorithm. An algorithm that tunes the communications must consider the following points:

- The publisher should send data in such a way that the subscriber never gets empty.
- The publisher should avoid filling its sending window, so it does not get blocked.

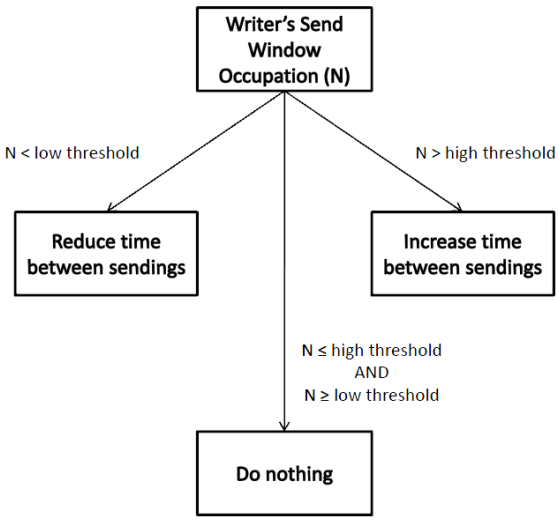

Fig 3. High level view of NAPA.

When the communication is reliable (by setting DDS QoS `reliable = reliability`), samples are kept in the sending window until the subscriber acknowledges them. This makes the sending window to be used as subscriber's feedback in the publisher's side. In terms of the publisher's sending window, according to the previous rationale:

- Sending window occupation should be highly enough to involve continuous transmission.
- Sending window should never get full.

These both are the two conditions that define the proposed algorithm behavior, hereafter referred to as Non-supervised Adaptative Publication Algorithm (NAPA).

Assuming the existence of an optimal local operation point, NAPA varies the time elapsed between consecutive data (T) as a function of the sending window occupation (N). As shown in Figure 3:

1. If N < `low_threshold`, decrease T.
2. If N > `high_threshold`, increase T.
3. If `low_threshold` ≤ N ≤ `high_threshold` do nothing.

Where `low_threshold`, and `high_threshold` (referred to as Acceptance Range) define the interval where the sending window occupation is expected to be in order to work in an optimal way.

To estimate the best values for the thresholds, we will follow two different approaches:

1. **Low_threshold**: The goal of this value is to modify the publisher's sending rate to have at least one unacknowledged sample in the sending window, so it does not get empty. For this reason, `low_threshold` is set equal to 1 for all the experiments in this paper.
2. **High_threshold**: The goal of this value is to reduce the publisher's sending rate when the publisher is being faster than the reception subscriber's capacity. To estimate the proper value for this variable, we are going to analyze the message passing diagram between publisher and subscriber.

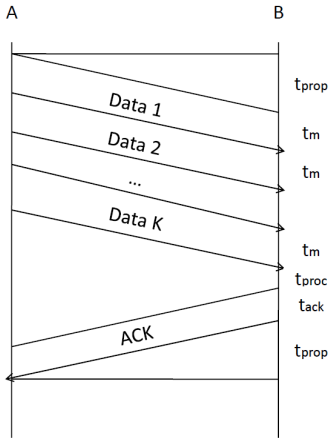

Fig 4. Time elapsed until first acknowledge message is received in the DDS Writer (A) when the DDS Reader (B) acknowledges samples in group of K samples.

Fig. 4 shows the elapsed time between different events that occur during the message passing. First, let´s estimate the size of the sending window (W) that allows the publisher to send data continuously:

$$W\ t_m\ \geq\ RTT_k$$

$$RTT_k = K\ t_m + 2\ t_{prop} + t_{proc} + t_{ack}$$

$$RTT_k = (K-1)t_m +\ RTT_1$$

$$W\ \geq \frac{RTT_k}{t_m} = \frac{RTT_1 C}{size_m} + (K-1)$$

where

- W: Sending window size.
- t_m: Time spent to put a message into the wire.
- RTT_k: Round-Trip Time measured when the subscriber needs to receive k messages to acknowledge.
- t_{prop}: Propagation time between publisher and subscriber.
- t_{proc}: Processing time in the subscriber.
- t_{ack}: Time spent to put an ack-message in the wire.
- C: Capacity of the network that connects publisher and subscriber.
- $size_m$: Size of the message.
- K: Number of messages that the subscriber needs to receive before sending an ACK. DDS defines this configuration parameter to save bandwidth.

After defining the minimum sending window size that is needed to have continuous transmission, `high_threshold` of the Acceptance Range must be determined. If we chose a value much higher than the defined by the above expressions, we would have bursts in the communication. In order to avoid that behavior, we chose the value defined by the following equation:

$$W = \frac{RTT_1 C}{size_m} + (K-1)$$

Given that the bandwidth, the message size and K are known values, only the RTT estimation is needed to estimate W. This can be statically done at the beginning of the communication or periodically in the transmission, what would modify the Acceptance Range and would get a better adaptation in dynamic operation scenarios.

IV. IMPLEMENTATION

For NAPA algorithm implementation, we adopted the following decisions:

- *RTT* is measured before the communication proceeds.
- The obtained value for the `high_threshold` (*W*) remains static for the session.
- To have full control, we need to spend time between sending consecutive samples without involving the CPU. To do this, elapsed time are expressed as the number of repetitions (`LoopCounter`) of a worthless operation. Time between consecutive samples will be modified indirectly just by increasing or decreasing `LoopCounter`. Note that this approach spends just a few microseconds in every execution, what provides a better granularity in comparison with the use of Sleep functions (which involves the CPU).
- The elapsed time between consecutive samples will not be modified every sent sample. Instead of that, we set an algorithm execution period.
- The elapsed time between consecutive samples will be modified as a function of the message size, a gain factor and a stability factor.
- Aiming at better adaptation speed as well as having higher precision, the gain factor will be dynamically adapted, starting at a high value and decreasing it with time.

Fig. 5 shows the NAPA implementation diagram flow:

1. At the beginning of the transmission, *RTT* is measured in order to estimate *W*.
2. NAPA modifies the `LoopCounter` once in each control period. During the transmission:
   a. If it is time for `LoopCounter` updating, compare the current sending window occupation with the Acceptance Range, and modify it consequently.
   b. If it is not time for updating it, do nothing.
3. Repeat `LoopCounter` times the worthless operations.
4. Send the message.
5. If we have sent all the messages, go to end. If not, back to step (2).

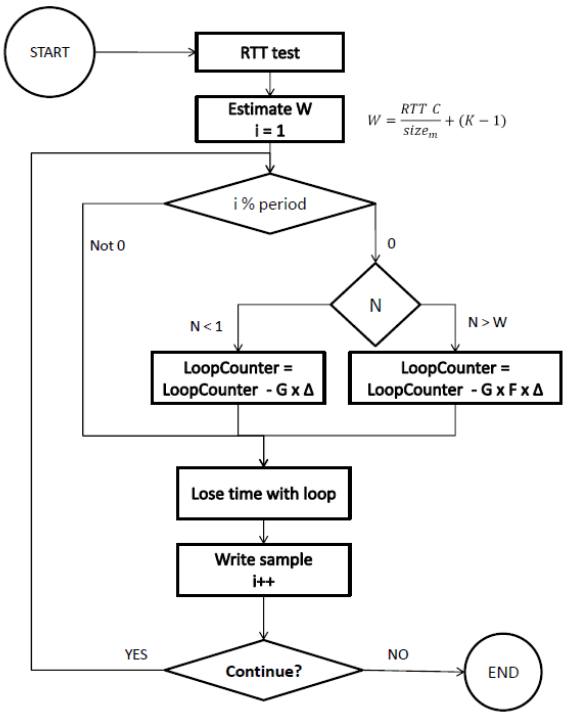

Fig 5. NAPA flow diagram.

We developed a NAPA prototype to evaluate our design and its performance improvement in comparison with non-using the proposed auto-tuning algorithm.

NAPA was implemented at the application level using the application RTIPerftest [2] as communications infrastructure. RTIPerftest is a tool used for analyzing communication performance within scenarios based in RTI Connext DDS [3].

RTIPerftest provides the features needed to measure latency and throughput, as well as the communication infrastructure needed to send data between two systems. There are several command-line options, including those to specify whether the application will act as the publisher or subscriber.

Several copies of the application can be executed (typically 1 publisher and 1 or more subscribers): The publisher application publishes throughput data and subscribes to latency echoes. The subscriber application subscribes to the throughput (in which the echo requests are embedded) and publishes the latency echoes (see Fig. 6).

Latency results are measured in the publisher and throughput results are estimated in the subscriber. RTIPerfTest includes many options to test almost every type of scenario. Among other it is possible to change data size, to specify QoS settings, different transport as well as disable acknowledgements, specify IP addresses for static discovery, activate batching, etc.

We adopted this tool for NAPA evaluation because of it could be easily done using the RTI Connext DDS API (specifically the C++ API).

Nevertheless, note that NAPA design is implementation-independent, as it only uses the sending window occupation as input to estimate the optimal sending speed. Consequently, NAPA can be implemented using any other DDS implementation.

RTIPerftest was modified to include NAPA functionality. It was compiled using gcc version 4.4.3 whereas the DDS implementation was RTI DDS 4.5d.

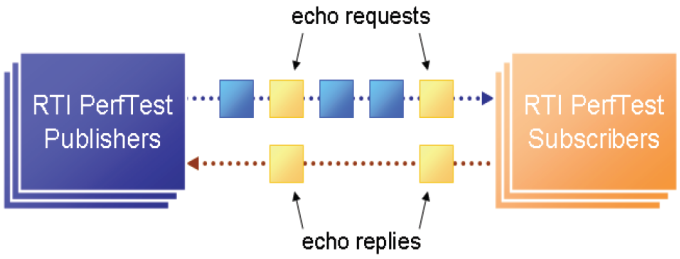

Fig 6. RTIPerfTest overview

V. EVALUATION

To evaluate the performance of NAPA, we made a set of experiments in a controlled LAN environment. Specifically, we measured the performance in terms of throughput and latency in a 1 to 1 reliable communication.

The evaluation environment was composed of two Core i5 at 2.66 GHz machines (lab01 and lab02) running Linux Kernel 2.6.32_22 x86_64 (Ubuntu 10.04) and the RTI DDS 4.5d middleware. These machines were connected by a 24-port Gigabit switch with VLAN support.

The experimental setup used in the evaluation of the NAPA algorithm were the following:

- Message size: from 5kB to 200kB.
- Reliable communication, one publisher to one subscriber via UDPv4.
- Gain factor starting at 1 and reduced to 0.2 after 30 seconds.
- Acknowledge (ACK) messages sent every 10 samples received.

Fig. 7 shows how NAPA improves the throughput for a wide range of message sizes. This is due to, as explained before, the number of retransmissions is reduced. The noticeable throughput reduction for using a data sizes slightly greater than 64kB is due to the size limit imposed by UDP packets: Note that form these message sizes it starts to use Asynchronous Writing in order to support bigger data sizes. NAPA reduces the performance loss when writing asynchronously and gets a better throughput for the rest of sizes.

Regarding the latency impact (Fig. 8), we see how one-way latency is decreased in almost an order of magnitude when applying NAPA. This reduction is the most important and noticeable result achieved by NAPA algorithm, because minimizing latency with a high level of throughput is considered one of the main goals of any real-time middleware.

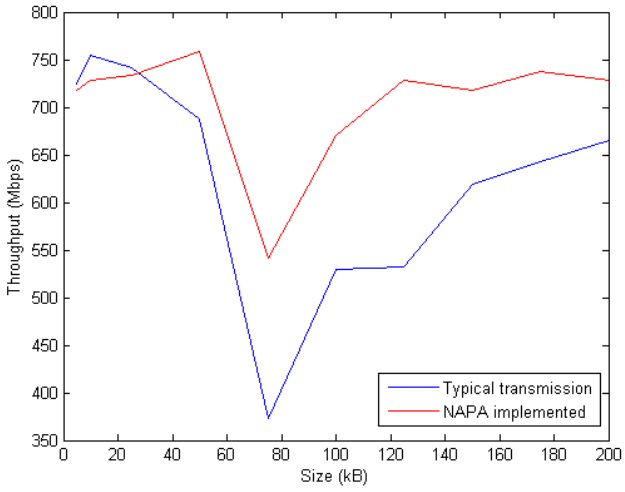

Fig 7. Throughput comparison between using and non-using NAPA.

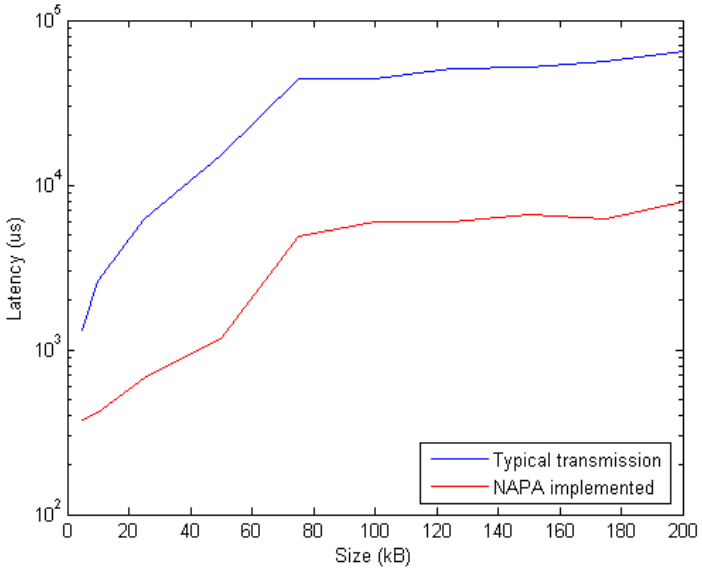

Fig 8. Latency comparison between using and non-using NAPA.

VI. CONCLUSIONS

In this paper we have presented NAPA, a dynamic auto-tuning algorithm for data-centric publish/subscribe systems.

Our algorithm is focused on avoiding publisher blocking and subscriber starvation problems by dynamically adjusting middleware parameters according to system conditions.

We have demonstrated that our algorithm effectively improves the performance of DDS-based systems both in terms of samples latency and overall throughput.

As future work, we are interested in including more dimensions in NAPA applicability:

- To apply NAPA in multicast scenarios. NAPA can be directly applied to 1 to many scenarios. DDS standard specifies that samples are removed from the send window in the publisher side when all the subscribers have acknowledged them. So, there won't be any significant change in the algorithm in order to be applied in multicast scenarios.
- To study the effect of applying NAPA in secure communications, where CPU usage is increased in the nodes due to secure protocols.
- To apply NAPA in disadvantage networks, that is in those with high loss communication. This investigation would show if NAPA improves performance when the nature of losses is not the communication congestion (overloading the subscriber writing faster than it can process).
- To include NAPA algorithm in the core DDS implementation to facilitate the auto-tuning t any application.

We are also interested in analyzing the performance improvement when increasing the NAPA complexity:

- By replacing the gain factor reduction model (linear) for a higher order model.
- By redesigning the "losing time between consecutive samples" method so it considers context switching. With the actual implementation, while the publisher application doesn't have the CPU the losing time loop doesn't advance, although in this cases time is going on what reduces its accuracy.

ACKNOWLEDGEMENTS

We would like to thank Gerardo Pardo-Castellote for his support and guidance during the investigation related to this tuning algorithm.

This research was partially founded by Spanish Ministry of Education (Collaboration Grant 2012-2013).

REFERENCES

[1] Data-Distribution Service for Real-Time Systems (DDS). v1.2. Tech. Rep., OMG. http://www.omg.org/cgi-bin/doc?formal/07-01-01.pdf.

[2] Real-Time Innovations. RTI Performance Test. Available at: http://na2.salesforce.com/_ui/selfservice/pkb/PublicKnowledgeSolution/d?orgId=00D3000000065k0&id=50140000000LQkK&retURL=%2Fsol%2Fpublic%2Fsolutionbrowser.jsp%3Fcid%3D02n400000004kPY%26orgId%3D00D3000000065k0&ps=1

[3] Real-Time Innovations. RTI Connext DDS. Available at: http://www.rti.com/products/dds/index.html

[4] OMG. Object Management Group, 2013.http://www.omg.org/

[5] Vladimir Dimitrov and Ventzislav Koptchev. PSIRP project – publish subscribe internet routing paradigm: new ideas for future internet. In Proceedingsof the 11th International Conference on Computer Systems and Technologiesand Workshop for PhD Students in Computing on International Conferenceon Computer Systems and Technologies, CompSysTech '10,pages 167–171, NewYork, NY, USA, 2010. ACM. ISBN 978-1-4503-0243 2. doi: 10.1145/1839379.1839409. http://dxhttp://dx.doi.org/10.1145/1839379.1839409

[6] PURSUIT. Publish Subscribe Internet Technology (PURSUIT) project site.Technical report, 2010. http://fp7pursuit.ipower.com/PursuitWeb/wp-content/uploads/2011/03/PURSUIT_fact_sheet.pdf

[7] Standard Performance Evaluation Corporation. Specjms2007 homepage, 2007. http://www.spec.org/jms2007/

[8] Securities Technology Analisis Center. Stac-m2 homepage. http://www.stacresearch.com/m2

[9] J. M. Lopez-Vega, J. Povedano-Molina, G. Pardo-Castellote, and J. M. Lopez-Soler, "A content-aware bridging service for publish/subscribe environments,"Journal of Systems and Software, vol. 86, no. 1, pp. 108-124, Jan. 2013. [Online]. http://dx.doi.org/10.1016/j.jss.2012.07.033

[10] J. M. Lopez-Soler, J. M. Lopez-Vega, J. Povedano-Molina, and J. J. Ramos-Munoz, "Performance evaluation of Publish/Subscribe middleware technologies for ATM (air traffic management) systems," in Workshop on Real-time, Embedded and Enterprise-Scale Time-Critical Systems, 2012.