Assessing data dissemination strategies within Triple Spaces on the Web of Things

Aitor Gómez-Goiri, Diego López-de-Ipiña
Deusto Institute of Technology - DeustoTech
University of Deusto
Avda. Universidades 24, 48007 Bilbao, Spain
{aitor.gomez,dipina}@deusto.es

***Abstract*—The upcoming of the Web of Things initiative has improved the integration of Internet-connected devices through the standard HTTP protocol and other web techniques. Unfortunately, it usually defines the data shared by these devices in a syntactic level, showing a lack of expressiveness. During the last decade, the Semantic Web (SW) has aimed to solve these problems by adding logic to the Web to make it machine-understandable and by therefore enhancing the interoperability of the applications using it. The SW is used in the Triple Space Computing paradigm, which proposes a blackboard model where semantically described knowledge is shared between different devices in a completely RESTful, and consequently WoT compliant, manner. This paradigm's shared blackboard can be implemented using many strategies, from centralized to completely distributed. In this work, we compare and analyze the behaviors of these two extreme cases in several simulations which try to represent common IoT scenarios. Finally, we propose an improvement of the completely distributed strategy by enabling the gossiping between devices.**

***Keywords*-Triple Space Computing; HTTP; Semantic Web; Internet of Things;**

I. Introduction

In recent years, the Web of Things (WoT) [1] has shown an emerging popularity for the integration of smart things. By embedding web servers into them to offer their capabilities using RESTful services, they can interoperate not only with other WoT devices, but also with other traditional web applications. As other RESTful compliant approaches, WoT usually represents these capabilities using user centered formats such as JSON and HTML, which purely describe the information at a syntactical level.

To overcome the limitations of this syntactic web a promising step would be to bring the WoT closer to the Semantic Web (SW) [2], which proposes to add logic to the web content. The SW aims to create a machine understandable web, where the synergy between agents not expressly designed to work together is promoted, improving the interoperability of the applications built on top of this enriched web.

Triple Space Computing proposes a blackboard model which uses the Semantic Web to describe the information shared. It has also been proved to be a completely RESTful approach [3] and therefore WoT compliant [4]. The way in which data is written and read in the space directly affects the performance of the solutions built on top of TS. In this work, we assess both distributed and centralized dissemination strategies for HTTP based TS running on IoT devices. In addition, we propose a gossiping based strategy which attempts to overcome their defects.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 describes Triple Space Computing. Section 4 presents the dissemination strategies. Section 5 describes how the experiments were conducted. Section 6 examines the results of the simulations. Finally, Section 7 concludes and outlines the future work.

II. Related work

In the WoT, different solutions have considered using semantics to enrich the data definition in a machine processable manner. Generally, these solutions embed the metadata in the HTML contents [5] returned by the Internet connected objects and are used to enhance the findability of the data by search engines, using microdata, microformats or RDFa. Other full semantics formats such as RDF or N3 can also be used to represent each of the resources shared by an object.

In any case, TS goes beyond simply returning semantic data in embedded web servers and proposes a blackboard model which access to a shared space, where the knowledge is read and written, using some coordination primitives. These primitives are RESTful compliant and therefore can be easily mapped into HTTP operations. This model offers a high degree of autonomy to the upper applications using it. Although, to the best of our knowledge no TS solution has been designed for the needs of the Internet of Things, other TS or tuple spaces semantic solutions have presented different strategies to distribute the space over the nodes in the most convenient way.

In centralized tuple spaces approaches we can find solutions which mainly focus on offering access to a space stored in a unique machine. This access is carried out by the clients using different methods which encapsulate the coordination primitives as in Semantic Web Spaces [6]. Remarkably, Smart M3 middleware proposes an architecture where the called Semantic Information Brokers (SIB) store

the information and perform reasoning process on behalf of the clients or Knowledge Processors (KP). Unfortunately, works where more than a SIB is used to store the information has not been presented yet, making it "de facto" centralized.

TripCom[1] uses a hybrid solution where information is distributed on an overlay network made up of different kernels using a Distributed Hash Table solution. Each kernel stores some triples and knows where the rest of them are stored by checking this routing table. The clients need to know one of these kernels to address their queries through it to the rest of the kernels. TripCom was conceived to store a huge amount of RDF triples taking special care of the scalability issues, but it was not designed to be run on devices with constrained resources.

Other tuple spaces solutions have used negative broadcasting strategy [7], [8], [9]. With this strategy, the writings are performed locally and the queries are broadcasted to all the reachable nodes which belong to the space. As can be seen, this strategy ensures that all the possible answers will be received, but at a highly resource consuming cost.

Finally, although they cannot be considered tuple space solutions, distributed semantic repositories (DSR) share some common data distribution problems with semantic tuple spaces. In these repositories, some applicable strategies such as flooding based [10], DHT based [11], overlay based [12], gossiping [13] and swarm-based [14], [15] have been presented. Unfortunately, the analysis carried out in the DSR field are not applicable since they diverge in the following aspects:

- The machines used in DSR are able to manage bigger amount of data than the resource constrained platforms used in IoT scenarios.
- The nature of the IoT nodes is more dynamic than the ones used in DSR and as a result they are less reliable.

So, considering all the strategies presented, how could we choose the most appropriate one for each Internet of Things environments? To answer the question, we present a thorough analysis in the following sections.

III. Triple Space Computing

The Triple Space Computing paradigm defines reading and writing primitives to access to a common space where semantic knowledge is held. By reading and writing in such a decoupled manner, three levels of autonomy are reached: location autonomy (information providers and consumers are independent from where the data is stored), reference autonomy (nodes do not need to know each other) and time autonomy (they communicate asynchronously). Additionally, the use of the RDF specification brings data schema autonomy, which makes the data independent of nodes' internal data schema. However, TS-based solutions are dependent on the ontology which defines the exchanged information or knowledge, so it is advisable the use or extension of standard or widely used ontologies in order not to isolate the application developed on top of TS.

[1]TripCom (IST-4-027324-STP, www.tripcom.org)

subject (URI | bnode)	**predicate** (URI)	**object** (URI | bnode | literal)
http://gomezgoiri.net/a	ontprefix:hates	http://triple-airlines.com
http://gomezgoiri.net/a	ontprefix:hasAge	"15"^^xsd:int
http://gomezgoiri.net/a	?p	?o

Figure 1. Basic composition of a RDF triple, two examples and a wildcard based template where both examples match. *ontprefix* is an alias, known as prefix in most Semantic languages, for the beginning of an URI.

The primitives defined in TS are *write*, *read*, *take* and *query*. The first one allows writing RDF Graphs, which is a set of RDF Triples (see Figure 1) identified by an URI, in a given space. The read primitive returns one of the graphs written into the space matching a template. The take primitive returns the same as the read, but extracting the graph from the space. Finally, the query returns all the RDF triples in a space which match a given template, no matter to which Graph they belong.

The template used in these primitives can be as simple as a triple pattern with wildcards (e.g. ?s) or as complex as an SPARQL query. In the simulations the first ones were used.

As was previously stated TS is completely compliant with the RESTful style. The resources (an space or a graph) can be accessed using their identifying URL and the appropriate HTTP verb, to create (HTTP POST/write), remove (HTTP DELETE/take) or retrieve (HTTP GET/read or query) semantic content. The use of HTTP brings huge benefits in terms of integration and adoption ease in new platforms.

Although in this work we focus on a key aspect of this implementation, i.e. how the data is spread among the different objects belonging to a space, it should not be forgotten the underlying HTTP TS implementation, which guides the design of the strategies and supports the simulation process.

IV. Data dissemination strategies

The tuple space-based computing approach consist of different elements (agents or nodes) which share information in a common space. This space can be centralized, substantially simplifying the application architecture and easing the reasoning process in semantic approaches.

Nevertheless, due to the distributed nature of the Internet of Things environments where the data is generated and consumed by many different devices, distributed approaches need to be considered. The peer-to-peer paradigm (P2P) is a prominent architecture to search for distributed data repositories which is usually divided in two main categories: unstructured and structured.

In structured P2P, a hash function is used both to determine where the content is stored and where to route a query.

It has not been considered for the analysis because it needs a costly process of reindexing terms whenever a peer joins or leaves the network, which may happen too frequently in IoT environments.

In unstructured P2P, each peer is connected to a limited number of peers, usually neighbors. The simplest unstructured approaches are the flooding-based ones, where both the queries (negative broadcasting) or the data to be queried (positive broadcasting) are spread among all the peers. To improve these approaches two common solutions can be used: group the nodes with similar interests (Semantic Overlay Networks) or allow them to know about the content shared by others (gossiping). We have selected the latter one to compare it with the negative broadcasting strategy.

In the following sections the following terms will be employed: the number of total requests (*r*), the period of time the strategy is evaluated (*t*), the given set of queries (*Q*, where each query is q_i), the writing frequency (w_f) and the nodes belonging to an space (*N*, where each node is n_i).

A. Centralized

The centralized strategy has a node which stores all the graphs periodically sent by the rest of the nodes. When a node wants to query the space, it directly addresses the query to this central node's web server. In sensor networks where data is continuously generated, the writing frequency of the sensed data is a vital parameter. With a high frequency the data stored in the server will be updated more frequently and therefore more accuracy will be achieved, but at the same time, the network overload will be severely affected. In other words, the writing frequency is a trade-off between the data freshness and the network overload (measured by the total requests):

$$r = |Q| + w_f.t.(|N| - 1) \tag{1}$$

Centralization has been traditionally used in IoT scenarios to prevent the objects from being directly requested and therefore save energy. Besides, it can be used in conjunction with a distributed strategy for those cases where devices cannot handle the Semantic Web (i.e. a fully IP-enabled gateway could represent several objects within its Zigbee network and yet communicate with other IP-enabled objects in a distributed fashion). In any case, we focus on maximizing embedded devices' capacities by using the Semantic Web, envisioning a world populated by autonomous devices which can cooperate between them. As a consequence, the dependency entailed by a unique central machine does not fit many of such scenarios.

B. Negative broadcasting

Negative broadcasting implies that all write operations are executed locally at the node, but all read and query operations are propagated to other nodes of a space. It suits perfectly to use cases where nodes create and manage their own information, such as a node in a mobile phone maintaining a user profile or embedded sensors managing their own generated data. In contrast with the centralized strategy, the HTTP requests generated do not depend on the writings, but it generates many request per query.

$$r = |Q|.(|N| - 1) \tag{2}$$

Although this strategy ensures that all the last available data will be retrieved, the main concern for its adoption in the IoT is that the network activity directly depends on the scale. As IoT scenarios are usually populated by a large number of nodes, each node will have to handle any request in the space no matter if it is relevant for them or not. As a result, the high network activity directly affects to the autonomy of the Internet-enabled devices.

C. Gossiping based strategy

Gossiping is used to improve broadcasting-based strategies, e.g. negative broadcasting, by reducing the amount of receivers for a given query requests. Ideally, the request should be transmitted only to the nodes which can answer something to the given request (i.e. the nodes having relevant content). Unfortunately, it is impossible to perfectly predict these nodes without having all the data they store locally available. As a result, the key for a successful gossiping strategy is to share just as much information as needed to predict with a admissible degree of accuracy and recall.

To deduce the information useful to predict the relevant nodes without sharing too much information between the nodes in TS, we have considered scenarios populated by mobile devices and sensor nodes as the ones which typically form IoT scenarios. Mobile devices normally share data like the user profile which is defined using a few ontologies to describe information such as users preferences which seldom change, whereas the sensors are constantly generating new instances of the same ontology. In both cases, the data shared by each node is described according to one or few vocabularies or taxonomies. At this point, is important to define the TBox and ABox following Nardi and Brachman's definition [16]. TBox contains the knowledge which describes general properties of concepts or terminology (e.g. the type of devices or the elements they have) and ABox contains knowledge that is specic to the individuals of the domain of discourse (e.g. the mobile brand is HTC or the sensed temperature is 3°C).

Considering spaces of nodes which know all the TBox used in the space is therefore plausible because it does not change too often and involves sharing much less information than the ABox. How the ontologies are obtained is not considered for the lack of simplification, but it will imply just a new request from the node which does not know a vocabulary to the one which knows it.

According to this situation, we have designed a strategy where the nodes gossip the classes of concepts (*rdf:type*)

shared by other nodes. With these information and the TBox, each node can check whether the information which matches certain templates is susceptible of being stored in other nodes. Another possibility would be to gossip the predicates, or the most popular subjects, but sharing the type of concepts and having the TBox information, these can be predicted with less accuracy, but also with less information being shared by the nodes (the amount of predicates is usually greater than the amount of classes in an ontology).

Given a template which has a predicate that according to the TBox relates a concept of the class A with another concept, the template will be transmitted to nodes having instances of the class A, as they are more likely to use this predicate in their graphs. For instance, consider that the following template is queried: *?s ssn:observes dbpedia:CO2*. Since according to the SSN ontology, which will be presented later on, only an instance of the class *ssn:Sensor observes* something, the querying node will send the template to the nodes which have instances of the class *ssn:Sensor*.

In addition, if the node can perform a reasoning process, unstated knowledge which can detect more relevant nodes can be inferred. For example, a node which has many instances of the class C, may also use the predicate if C is a subclass of A. Thanks to the reasoning, we can discover that the node has knowledge of the type A and therefore may be relevant (in the example, instances of *ssn:Sensor* class' subsclasses such as *Accelerometer*).

So, if we refer to the algorithm shown in the Algorithm 1 as the n_r function, using it and the TBox information, we have the following equation to calculate the total amount of HTTP requests needed. In the worst case scenario, the function will return n-1 and we will have the negative broadcasting case. The problem has been simplified by considering that the TBox information does not change and therefore the gossiping request will only be needed the first time each node initiates a query. Each node will need to ask to the rest of the nodes ($|N| - 1$) only if it tries to initiate a query (i.e. at maximum $|N|$ nodes will initiate this process, having $|N|.|N - 1|$ as the worst case for g).

$$r = \sum n_r(q_i) + g \tag{3}$$

V. Simulations

We opted for a simulation study to compare the performance of the strategies described in the previous section. We anticipated to see poor scalability of the Negative Broadcasting and Centralized strategies for query and write intensive scenarios respectively. Moreover, we expected a substantial improvement by complementing Negative Broadcasting with a Gossiping mechanism which could refine which nodes should be queried.

Algorithm 1 Algorithm to deduce the list of relevant nodes

```
ungossiped = getNotGossipedNodes()
if len(ungossiped)>0:
  for uni in ungossiped:
    # ask to the "uni" node the types (classes)
    # of instances it has
    gossiped_base[uni] = gossipClassesFrom( uni )


relevant_nodes = []
for ni in N:
  if template.object in gossiped_base[ni]:
    relevant_nodes.append( ni )
  elif template.predicate != WILDCARD:
    # if according to TBox and gossiped_base
    # "ni" has instances of any class in the
    # range of template.predicate's adds it

    pr = range(tbox, template.predicate)

    if gossiped_base[ni].hasInstancesOfClass( pr ):
      relevant_nodes.append( ni )
  else:
    # we have not enough information to decide
    # whether the node is not relevant
    relevant_nodes.append( ni )
```

Table I
Configuration Parameters.

Name	Description
Network Size	The number of nodes in a network.
Number of writes	Amount of write primitives performed during the simulation period.
Number of queries	Amount of query, read or take primitives performed during the simulation period.

A. Methodology

By varying the configuration parameters presented in the Table I a wide range of scenarios could be simulated. As we were assessing objects with IP connectivity, the topology considerations have been omitted.

Since we wanted to simulate the Triple Space Computing paradigm over HTTP, the communication between the nodes was point to point and the data exchanged were RDF Triples. The node discovery process was ignored since it will show similar additional overhead for each strategy. It can be considered transversal to what was being measured.

To represent the data managed by each node, we first considered using LUBM[2], a synthetic benchmark. Unfortunately, it creates instances from the same few classes for each node, making all the nodes have the same TBox. In our opinion, this does not represent faithfully Internet of Things scenarios where there exist heterogeneous devices which therefore share disparate information.

In our second attempt we sought a dataset which could represent this heterogeneity. The *Semantic Sensor Network Ontology* (SSN)[3] was created by the W3C Semantic Sensor Network Incubator Group to represent diverse sensor net-

[2]http://swat.cse.lehigh.edu/projects/lubm/

[3]http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/

Table II
CHARACTERISTICS OF THE DEVICES USED IN THE EVALUATION AND THE TIME NEEDED BY THEM TO RESPONSE TO CONCURRENT HTTP REQUESTS (MILLISECONDS).

	XBee Gateway		**FoxG20**	
Processor	-		400Mhz Atmel ARM9	
RAM	8 MB		64 MB	
Concurrent requests	Mean	Std dev (σ)	Mean	Std dev (σ)
1	77	1	17	0
5	392	8	97	16
10	775	8	174	28
15	-	-	282	43
20	-	-	375	30
25	-	-	460	30
30	-	-	540	35
35	-	-	632	29

work environments. The ontology has been used in many different projects and scenarios to semantically describe the data provided by heterogeneous sensors. Specifically, we used the two publicly available meteorology and environmental quality datasets based on the SSN ontology, provided by the University of Utah[4] and by the University of Deusto[5] respectively. These datasets contain descriptions about the sensing stations and the data sensed by them during certain periods, the analogy between meteorology stations which have different sensors and the IoT devices is reasonable. The dataset has been adapted to provide just one measure of each sensor at each moment (to emulate the storage restrictions from embedded devices) and to use as many stations as nodes has the network (depending on the network size).

We simulated the environments using SimPy[6], a process-based discrete-event simulation language for Python. To accurately simulate the time needed by each node to provide a response, we considered measures taken from real embedded web servers running on an IP gateway[7] for XBee sensors'[8] and on a FoxG20[9] [17]. Table II shows the measures used for the parametrization.

B. Performance Metrics

We evaluated the three strategies selected against each other regarding the metrics described below. These metrics are intended to capture the fundamental properties relevant to this comparison.

- *Precision*: the fraction of nodes which answered relevant results (those responses which were not *not found* responses).

[4]http://wiki.knoesis.org/index.php/LinkedSensorData
[5]http://dev.morelab.deusto.es/bizkaisense
[6]http://simpy.sourceforge.net
[7]http://tinyurl.com/connectportx2
[8]http://tinyurl.com/xbee-sensors
[9]http://www.acmesystems.it

Table III
TEMPLATES USED IN THE EVALUATION OF THE GOSSIPING APPROACH.

Name	Template
t1	?s rdf:type ssn-weather:RainfallObservation
t2	?s rdf:type ssn:Observation
t3	?s ssn:hasLocation ?p
t4	?s wsg84:long ?p
t5	?s ssn:observedProperty ?o
t6	?s smart-knife:hasMeasurementPropertyValue ?o
t7	bizkaisense:ABANTO ?p ?o

- *Recall*: the fraction of relevant answers that are returned.
- *Response time*: the average response time needed to obtain the response to a query, read or take primitive.

VI. RESULTS

A. Gossiping approach

The precision and recall of the gossiping algorithm has been evaluated for a network size of 159 nodes issuing the query template shown in the Table III. In average, the nodes have instances belonging to 6 different classes (σ=4) from a total 87 distinct classes in the space. When these classes are expanded with a reasoning process, the nodes store 20 different concepts from a total of 113 classes in the space (σ=4).

The Table IV shows the recall and precision obtained with the different templates. The first template shows a perfect prediction for these classes detected in the instances of the TS. The second shows how if the data consumer expands the gossips through a inference process but the data provider does not reason over its content, the provider is not going to be able to answer a concept it implicitly has. From the third to the sixth template, they define a predicate whose range is not explicitly stated in the gossiping. When the gossiping is expanded, we obtain high precision for the first ones, with a great recall in the fifth one. The sixth one, returns two possible nodes while just one holds a triple using this predicate. Finally, the seventh template shows the main limitation of the proposed template: a low precision for a template where no predicate is defined. This is due to the fact that with the TBox information gossiped ABox information cannot be inferred. To overcome this limitation the most popular and long-lasting URIs could be also transmitted during the gossiping process.

At this point, it is worth noticing the overhead that the reasoning process can introduce in resource constrained devices. While the XBee gateway is not able to reason, the FoxG20 took about 50 seconds to load the TBox information from SSN and about 1.5 seconds to load each RDF graph with one measure extracted from the dataset used. The energy consumed during the reasoning process is similar to the consumption measured during a period with continuous network activity (see Figure 4).

Table IV
GOSSIPING ALGORITHM'S PRECISION AND RECALL.

	No inference		Inference	
Template	**Precision**	**Recall**	**Precision**	**Recall**
t1	1.0	1.0	1.0	1.0
t2	1.0	1.0	0.0	0.0
t3	1.0	1.0	1.0	1.0
t4	0.0	0.0	1.0	0.647
t5	0.0	0.0	1.0	0.99
t6	0.0	0.0	0.5	1.0
t7	0.006	1.0	0.006	1.0

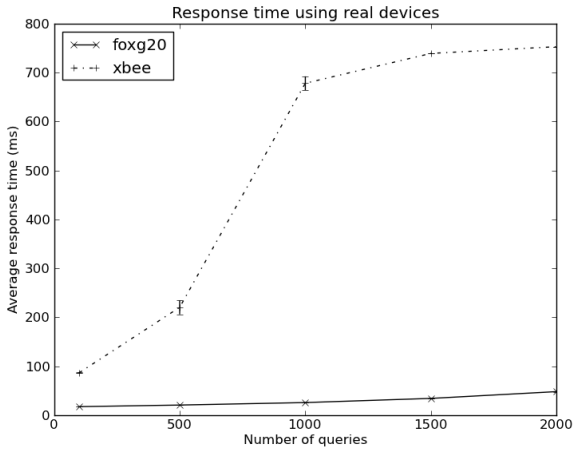

Figure 2. Overhead generated by negative broadcasting in real devices for different amount of queries in one minute.

Fortunately, the TBox loading needs to be done just once and the reasoning over the gossiped information can be done before the first TS primitive is spread among the nodes. The devices unable to reason could use an external reasoning service to obtain the expanded gossiping even if they are unlikely to initiate queries.

B. Strategies performance in real devices

This experiment simulated a network of 100 nodes populated by nodes parametrized to behave as XBee gateways or FoxG20 embedded devices (see Table II).

The response time (see Figure 2) exposes a worse performance for the XBee, which in any case can be able to answer more than 1000 queries per minute before it starts rejecting responses. This upper limit makes feasible most of the scenarios which could be conceived with them.

C. Network overload

An elemental mechanism to increase the autonomy of the devices is to reduce their activity by minimizing the number of request they have to handle. This is graphically represented for the FoxG20 platform by the Figure 4, where the energy consumption jumps to 25% when both it reasons or it handles multiple requests.

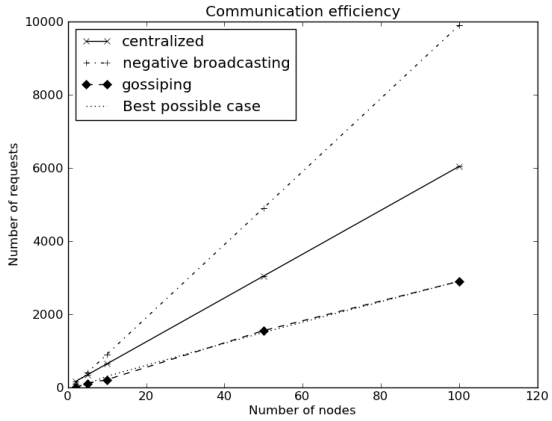

Figure 3. Overhead generated by the different strategies measured in the number of HTTP request needed to answer a query.

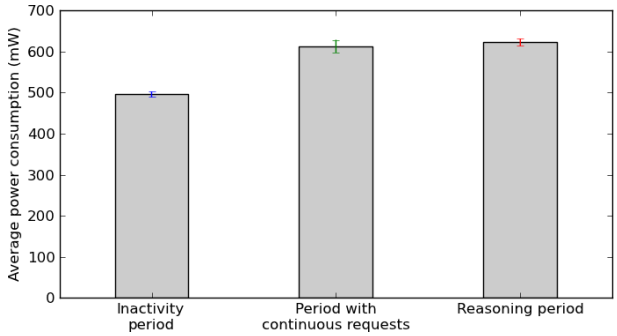

Figure 4. Energy consumption for FoxG20 during different activity periods (miliwatts).

To measure the variations in the system load as the network size increases, we generated networks of 2, 10, 50 and 100 nodes. The simulation lasted one minute (t=60000) and had a writing frequency of one second (w_f=1000) and 100 queries which employed the template *t1*.

In the Figure 3 a linear increment is appreciated as the network size increases for all the strategies. The centralized strategies directly depend on the writing frequency which can drop depending on the necessity of freshness in the scenario. The negative broadcasting and the gossiping strategy on the contrary, are proportional to the number of queries in the given time. Even if 100 queries may be too many for a minute in some scenarios, the figure shows how the negative broadcasting slope is the most pronounced one, being the worst strategy even for just few nodes. The gossiping strategy considerably reduces the HTTP requests needed by the broadcasting-based strategy. In the Figure 3, this reduction adjusts to the best possible case, but depending on the precision of the algorithm for the given query, it can be situated anywhere between the negative broadcasting and the best possible case.

VII. Conclusion

This paper assesses the adoption of different data dissemination strategies within an HTTP based Triple Space Computing solution. Centralized and fully distributed strategies have confirmed the poor scalability predicted. These cases can be improved using a gossiping strategy which shares the classes of the instances hold by each node of the space.

The simulations carried out together with the response times obtained from real devices confirm the need of the gossiping approach since the negative broadcasting quickly floods the most simple ones. This behavior is particularly harmful for IoT devices, since their energy consumption rises when requests are handled.

For our future work, we aim to refine the proposed gossiping strategy to increase the precision for those cases where no predicate is defined in the template. Furthermore, apart from using real datasets and response times from real IoT devices in the simulations, specific scenarios from the literature could be simulated to obtain more accurate performance results.

Acknowledgment

This work has been supported by research grants TIN2010-20510-C04-03 (TALIS+ENGINE project), funded by the Spanish Ministry of Science and Innovation, and IE11-316 (FUTURE INTERNET II project), funded by the Basque Government, ETORTEK 2011.

References

[1] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the internet of things to the web of things: Resource oriented architecture and best practices," in *Architecting the Internet of Things*. Springer, May 2011.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, no. 5, p. 3443, 2001.

[3] A. G. Hernández and M. N. M. García, "A formal definition of RESTful semantic web services," in *Proceedings of the First International Workshop on RESTful Design*. New York, NY, USA: ACM, 2010, p. 3945.

[4] A. Gómez-Goiri and D. López-de-Ipiña, "On the complementarity of triple spaces and the web of things," in *Proceedings of the Second International Workshop on Web of Things*. New York, NY, USA: ACM, 2011, p. 12:112:6.

[5] S. Mayer and D. Guinard, "An extensible discovery service for smart things," in *Proceedings of the 2nd International Workshop on the Web of Things*. San Francisco, CA, USA: ACM, Jun. 2011.

[6] L. Nixon, O. Antonechko, and R. Tolksdorf, "Towards semantic tuplespace computing: the semantic web spaces system," in *Proceedings of the 2007 ACM symposium on Applied Computing*, 2007, p. 360365.

[7] R. Krummenacher, D. Blunder, E. Simperl, and M. Fried, "An open distributed middleware for the semantic web," *International Conference on Semantic Systems (I-SEMANTICS)*, 2009.

[8] A. Murphy and G. Picco, "Transiently shared tuple spaces for sensor networks," in *Proc. of the Euro-American Workshop on Middleware for Sensor Networks*, 2006.

[9] A. Gómez-Goiri, M. Emaldi, and D. López-de-Ipiña, "A semantic resource oriented middleware for pervasive environments," *UPGRADE journal*, vol. 2011, Issue No. 1, pp. 5–16, Feb. 2011.

[10] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov, "Piazza: Data management infrastructure for semantic web applications," in *Proceedings of the 12th International Conference on World Wide Web*, 2003, pp. 556–567.

[11] M. Cai and M. Frank, "RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network," in *Proceedings of the 13th International Conference on World Wide Web*. New York, NY, USA: ACM, 2004, p. 650657.

[12] C. Doulkeridis, K. Norvag, and M. Vazirgiannis, "DESENT: decentralized and distributed semantic overlay generation in P2P networks," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 1, p. 2534, 2007.

[13] J. Zhou, W. Hall, and D. D. Roure, "Building a distributed infrastructure for scalable triple stores," *Journal of Computer Science and Technology*, vol. 24, no. 3, pp. 447–462, 2009.

[14] H. Mhleisen, A. Augustin, T. Walther, M. Harasic, K. Teymourian, and R. Tolksdorf, "A Self-Organized semantic storage service," 2010.

[15] P. Obermeier, A. Augustin, and R. Tolksdorf, "Towards swarm-based federated web knowledgebases," *Electronic Communications of the EASST*, vol. 37, no. 0, 2011.

[16] D. Nardi and R. Brachman, "An introduction to description logics," *The description logic handbook: theory, implementation, and applications*, pp. 1–40, 2003.

[17] A. Gómez-Goiri, P. Orduña, D. Ausín, M. Emaldi, and D. López-de-Ipiña, "Collaboration of sensors and actuators through triple spaces," in *IEEE Sensors 2011*, Limerick, Ireland, 2011.