Bloom Filter Based Discovery Protocol for DDS Middleware

Javier Sanchez-Monedero[a,*], Javier Povedano-Molina[b], Jose M. Lopez-Vega[b], Juan M. Lopez-Soler[b]

[a] *Department of Computer Science and Numerical Analysis, University of Córdoba, Rabanales Campus, Albert Einstein building 3[rd] floor, 14071, Córdoba, Spain*
[b] *Department of Signal Theory, Telematics and Communications, University of Granada, ETSI Informática y de Telecomunicación, C/ Periodista Daniel Saucedo Aranda, s/n, 18071 Granada, Spain*

Abstract

The *Data Distribution Service* (DDS) middleware has recently been standardized by the OMG. Prior to data communication, a discovery protocol had to locate and obtain remote DDS entities and their attributes. Specifically, DDS discovery matches the *DataWriters* (DWs) and *DataReaders* (DRs) entities (*Endpoints*) situated in different network nodes. DDS specification does not specify how this discovery is translated "into the wire". To provide interoperability and transparency between different DDS implementations, the OMG has standardized the DDS Interoperability Wire Protocol (DDS-RTPS). Any compliant DDS-RTPS implementation must support at least the SDP (Simple Discovery Protocol). The SDP works in relatively small or medium networks but it may not scale as the number of DDS *Endpoints* increases. This paper addresses the design and evaluation of an SDP alternative – which uses Bloom Filters (BF) – that increases DDS scalability. BFs use Hash functions for space-efficient probabilistic data set representation. We provide both analytical and experimental studies. Results show that our approach can improve the discovery process (in terms of network load and node resource consumption), especially in those scenarios with large *Endpoint* per *Participant* ratios.

Keywords: DDS, Data Distribution Service, discovery, middleware, bloom filter, peer-to-peer, RTPS

*Corresponding author.
Email addresses: `jsanchezm@uco.es` (Javier Sanchez-Monedero), `jpovedano@ugr.es` (Javier Povedano-Molina), `jmlvega@ugr.es` (Jose M. Lopez-Vega), `juanma@ugr.es` (Juan M. Lopez-Soler)

1. Introduction

The *Data Distribution Service* (DDS) [24, 25] is high performance middleware for predictable distribution of data with minimal overhead. DDS has been standardized by the Object Management Group (OMG) to expedite publish/subscribe communications in real-time and embedded systems. OMG DDS specification is increasingly being used to integrate real-world systems. Examples include Air-traffic Control Systems, Navy Combat Management Systems, Automatic Stock Trading Systems, as well as Industrial control and SCADA systems. The Data Distribution Service implements a true distributed peer-to-peer architecture that exploits a data-centric approach leveraging reliable and efficient end-to-end communications. The DDS data-centric approach facilitates the interoperability of heterogeneous systems by building a Global Data Space (GDS). Some applications publish data in the GDS and, in like manner, others use the data space to subscribe to information of interest.

A key feature of DDS architecture is that information consumers and producers are decoupled in *space* (providers and consumers can be located anywhere), in *time* (there is no need of simultaneous end-point availability) and in *platform* (providers and consumers can be developed in diverse operating systems, hardware architectures and languages). DDS also achieves *multiplicity* decoupling, *i.e.* the middleware manages multiple simultaneous sources and destinations for the same data. Summing up, DDS sets up an overall decoupled data-centric publication-subscription paradigm.

To join the GDS, the middleware must be able to locate both remote *Publisher* and *Subscriber* entities to obtain their attributes and accordingly, to communicate with them. In doing so, a discovery protocol is involved. Discovery is a time-consuming process that might run in scenarios with scarce resources (such as memory and network bandwidth). Due to these reasons it is important to provide efficient discovery protocols as this helps to improve DDS scalability.

Different discovery protocols have been proposed in assorted contexts. For example, in wired networks Jini [33], the IETF Service Location Protocol [11] and Universal Plug and Play [8] have been asserted. In wireless networks, among many others works, both single [13] and multi hop [20] discovery schemes have been proposed. More discovery schemes are briefly described in Section 7.

The DDS standard specifies discovery information exchanged between publishers and subscribers. However, the standard does not specify how this discovery information is physically sent through the network. The lack of a DDS wire protocol has an immediate consequence: different DDS implementations are not necessarily interoperable. In order to address this situation, the OMG has standardized the DDS Interoperability Wire Protocol (DDS-RTPS) [25]. DDS-RTPS relies on the *Real-Time Publish-Subscribe protocol* (RTPS) [25] to transmit data over the network. RTPS defines the SDP (Simple Discovery Protocol) to be used for DDS entities in passive searching. The discovery protocol defines a meta-traffic exchange that enables DDS entities to identify, locate and obtain attributes of all the other GDS entities. Thereby, DDS based applications automatically obtain a complete picture of their *Domains*.

For the purpose of interoperability, any compliant DDS-RTPS implementation must provide at least the SDP discovery protocol. Originally, SDP was specified for relatively small or medium networks with a relatively stable GDS (that is, low birth and death *Endpoints* rates). However, it may not scale as the number of DDS *Endpoints* increases.

The following example highlights some SDP limitations. Let's imagine a unicast scenario with 100 *Domain Participants* and 2000 *Endpoints*. During the discovery phase, every *Participant* will send and receive approximately 4000 SDP messages to discover all the other *Participants* and *Endpoints* in the domain (see Eq. (1) in Section 3.2). In this scenario, the total of messages sent through the network will be approximately equal to 200000 (see Eq. (2)), whereas not all of them will be necessary given that not every *Endpoint* will be interested in discovering every other *Endpoint*. For example, in a typical sensor network like a naval frigate [12] most of the sensor publishers (temperature, radars, etc.) will only be interested in discovering other subscriber entities. However, they will not be interested in other sensor publishers. If every *Endpoint* were interested in just 50% of the *Endpoints*, half of the SDP network load would be wasted.

This paper proposes an SDP enhancement. Broadly speaking, our goal is to improve discovery protocol scalability, and more precisely to reduce the network load and memory requirements in SDP while preserving both the DDS decoupled publication-subscription GDS model and its peer-to-peer nature.

We harness the power of Bloom Filters (BF) [2] in SDP to improve DDS scalability. Inspired by its traditional use in data base query and more recently in some network applications [4], we propose to include BF in SDP –hereafter referred to as *SDPBloom*–. Bloom Filters, originally conceived by Burton H. Bloom in the 70s [2], are space-efficient probabilistic data structures that were defined for efficient membership queries.

Basically, the main idea behind *SDPBloom* is for each DDS *Participant* to summarize and send all its *Endpoints* information with a BF. With this simple approach, any *Participant* in the GDS will efficiently receive the whole remote *Participant* discovery-related information.

As we will show, in *SDPBloom* the number of sent messages is not dependent on the *Endpoints* number (E). More precisely, it approximates $P \cdot P$, where P is the number of *Participants*. In the baseline SDP, however, this number is equal to $P \cdot E$. Therefore, given that usually $P < E$, our approach improves discovery performance. In the following sections, analytical and experimental results demonstrate the benefits of *SDPBloom*.

The rest of the paper is organized as follows: Section 2 introduces DDS terminology and basic concepts; Section 3 analyzes the DDS Simple Discovery Protocol; the next section provides basic Bloom Filters background; Section 5 describes the proposed discovery solution and compares it to the SDP baseline scheme; Section 7 reviews some related works; Section 6 reports on the experimental tests which complete our analytical study; finally, Section 8 summarizes the conclusions and possible extensions to this work.

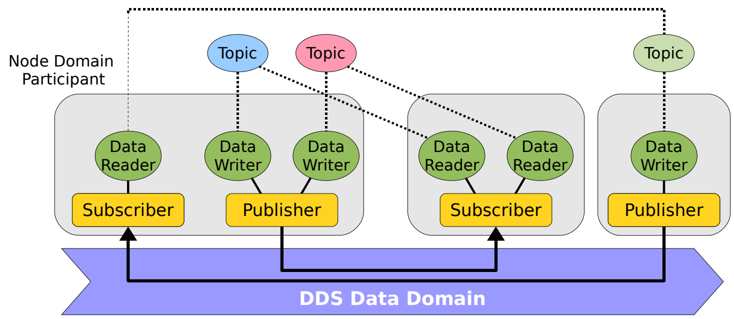


Figure 1: DDS entities relationship.

2. The Data Distribution Service

The OMG DDS standard is specified in one main document and several supplemental ones. The main document is the Data Distribution Service for Real-time Systems specification [24], and it defines the publish/subscribe communications model (APIs, Semantics, Quality of Service, Programming Model, etc) for distributed systems. It also includes the Data-Centric Publish-Subscribe (DCPS) communication standard. The DCPS conceptual model is based on the abstraction of a Global Data Space. *Publisher* applications post data into the GDS, and DDS middleware efficiently disseminates them to all interested *Subscriber* applications with a high level of transparency. Therefore, DDS middleware decouples the production and consumption of information through the GDS. Additionally, the interoperability of different applications is also enhanced given that the GDS provides a framework for flexible and transparent data sharing. Closely related, the OMG DDS Interoperability Wire Protocol (DDS-RTPS) specification [25] (based on RTPS) enables different DDS implementations to be inter-operable.

For better understanding, the following DDS concepts are briefly stated – some of them are depicted in Fig. 1–. In some cases, we also provide illustrative examples applied to the Naval Frigate [12] use case:

Domain. It is a virtual network concept which helps to isolate and optimize communications among distributed applications that share common interests. The DDS applications are able to publish and subscribe data if they belong to the same *Domain*. The *Domain* in the Naval Frigate example involves the GDS in which different elements (radars, sensors, Integrated Machinery Control Systems, weapon systems, workstations, Combat Information Centers, staff rooms, comms room equipment, etc.) produce and consume information.

Domain Participant. (Or simply *Participant*) It represents the application involvement in the communication plane in a given *Domain*. It isolates applications running on the same set of physical computers. A *Participant* operates as a service entry-point and behaves as a container for other entity objects as well. In the example, any radar, sensor, Integrated Machinery Control System, etc. access to the GDS through the corresponding *Participant*.

Topic. It materializes the interaction between the GDS and the applications. A *Topic* can be defined as the logical channel that associates *Participants*. It is identified by its unique name in the whole *Domain*. It fully specifies the type of data that can be communicated when publishing or subscribing information to the DDS Global Data-Space. Examples of Topics could be the radar data, GPS position, etc.

Publisher. It is the object responsible for the actual data dissemination. It may publish data objects of different types by using different *DataWriters* (see below). In the example, Publishers are any of the elements that produce information, for instance, a radar or GPS system.

Subscriber. A *Subscriber* is an entity responsible for receiving published data. It provides the received data to the application. A *Subscriber* reads *Topics* in the GDS for which a matching subscription exists and informs the *DataReaders* (defined below) that data have been received. In the naval frigate example, any of the elements that consume information, such as the Combat Information Center, are Subscribers.

DataWriter. Applications use *DataWriters* to write data in the GDS of a *Domain* through a *Publisher*. A *DataWriter* acts as a typed accessor to a *Publisher*. *Typed* means that each *DataWriter* object is dedicated to one application data type (i.e. GPS data).

DataReader. It notifies an application that data from the GDS are available. For accessing received data, the application must use a typed *DataReader* attached to the *Subscriber*.

Quality of Service (QoS). The DDS QoS is a set of data transmission policies that not only control the use of resources such as network bandwidth, memory, processor usage, etc. but also define *Topic* properties such as data *persistence*, *reliability*, *timeliness*, etc. QoS politics customize the DDS service provided for application requirements.

DDS can be described as an overlay peer-to-peer structure where the *Publishers* of a given *Topic* are linked to all *Subscribers* for the same *Topic* through the GDS. Hereafter, *DataReaders* and *DataWriters* will be jointly referred to as *Endpoints*.

To communicate *Publishers* and *Subscribers*, DDS relies on a discovery protocol which allows a *Publisher* to dynamically discover compatible *Subscribers* and vice-versa. Any OMG DDS-RTPS standard compliant implementation needs to provide at least the SDP discovery protocol to identify the presence or absence of other *Endpoints* when they either join or leave the *Domain*. The discovery protocol accomplishes the transparent and inter-operable plug-and-play dissemination of all the information between *Publishers* and *Subscribers*.

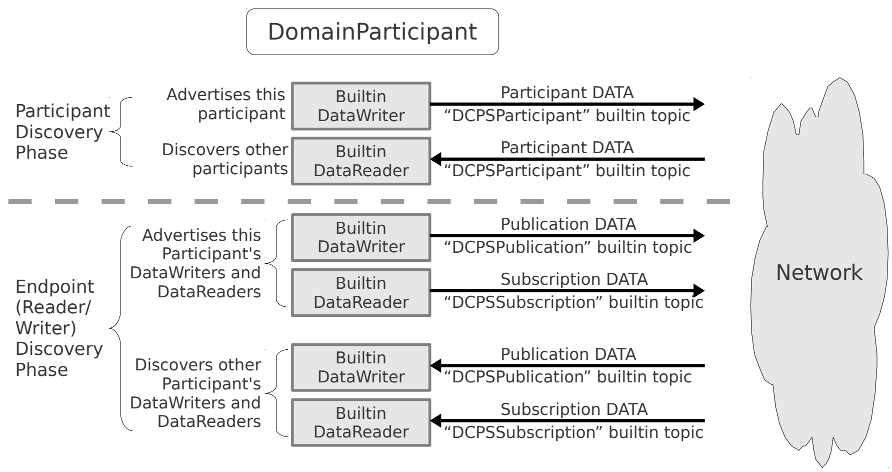


Figure 2: DCPS built-in entities for discovery purposes.

3. DDS Simple Discovery Protocol

According to the DDS Interoperability Protocol [25], any discovery protocol must be divided into two consecutive phases: the *Participant Discovery Protocol* (PDP) and the *Endpoint Discovery Protocol* (EDP). The purpose of PDP is to discover new *Participants* in the *Domain*. Whenever a new *Participant* is discovered, the EDP procedure is triggered to exchange local and remote *Endpoints* information between two *Participants*. Different implementations may choose to support multiple PDPs and EDPs, possibly vendor-specific. As long as two Participants have at least one PDP and EDP in common, they can exchange the required discovery information. For the purpose of interoperability, at least the Simple Discovery Protocol –explained in the following subsection– must be supported.

3.1. SDP description

Simple Discovery Protocol is divided into the *Simple Participant Discovery Protocol* (SPDP) and the *Simple Endpoint Discovery Protocol* (SEDP).

SDP utilizes DDS publications themselves for discovery purposes. It uses a special set of *Topics*, *DataReaders* and *DataWriters* for advertising and discovering other *Participants* and *Endpoints*. These special entities are called *built-in entities*. To improve performance, the discovery process can be tuned with specific QoS policies which could be applied to the built-in entities.

Fig. 2 shows the *Topics* related to SPDP ("DCPSParticipant") and SEDP ("DCPSSubscription", "DCPSPublication"). For each one of these *Topics* there is a specific associated data type. For example the *SPDPdiscoveredParticipantData* is the data type used in the "DCPSParticipant" *Topic*.

Bootstrapping– The discovery process is started from a list of known hosts. It contains the locators (typically unicast or multicast IP addresses) for which a *Participant* will announce its presence. Alternatively, if there are no specified IP addresses, default addresses will be used. Both options can be used together.

When a *Participant* in a node is enabled, the first discovery stage consists in discovering other *Participants*. This discovery –restricted to *Participants* in the same DDS *Domain*– is done via PDP. In SPDP, a special message called *SPDPdiscoveredParticipantData* or simply *Participant DATA*, is periodically sent to known peers when a *DomainParticipant* is created or deleted. If multicast is available, a unique *Participant DATA* message is sent by each *Participant*.

Participant Announcement– By default, when new *SPDPdiscoveredParticipantData* messages are received, the *SPDPdiscoveredParticipantData* itself is sent to the remote *Participant*. Then, the remote *Participant* is stored locally.

The *SPDPdiscoveredParticipantData* contains information for establishing communication between two *Participants*, that is, information related to the protocol version, vendor identification, unicast and multicast locators (transport protocol to use, IP address and port combinations) and information about how to track *Participant* liveliness. Also, the information contained includes which Endpoint Discovery Protocols the remote *Participant* supports. Therefore, the proper Endpoint Discovery Protocol can be selected to exchange *Endpoint* information with the remote *Participant*.

Endpoint Announcement– Similarly, SEDP publication and subscription information is composed of the data needed for matching local and remote *Endpoints*. These data are specified as:

1. The *Topic* name of the *Endpoint*.
2. The data *type name*.
3. The data *typecode*. It is defined as the data-type structure description for a DDS object.
4. The supported QoS parameters such as accepted deadline, reliability level, etc.

The DDS middleware must check that the previous first three data –*Topic*, data *type names* and *typecode*– are the same. It must also verify that the offered and requested QoS parameters are compatible. If that is the case, the remote *Endpoint* is suitable for starting publication-subscription communication. Even though the *typecode* is not included in standard discovery information, the data *type* description is usually included in DDS implementations for proper data serializing, de-serializing and error checking purposes. For example, a publication *Topic* name and *type name* can match another subscription but if the *typecode* is not exactly the same, the communication will not be established in order to preserve data correctness.

Due to the pure peer-to-peer nature of RTPS/DDS, each *Participant* stores the information about discovered *Participants*, associated publications and subscriptions in a local database. The SEDP protocol sends the *Endpoints* information for every *Participant* in the local *Participant* database. Therefore, each *Participant* receives all the discovered *Participants' Endpoints* information. For

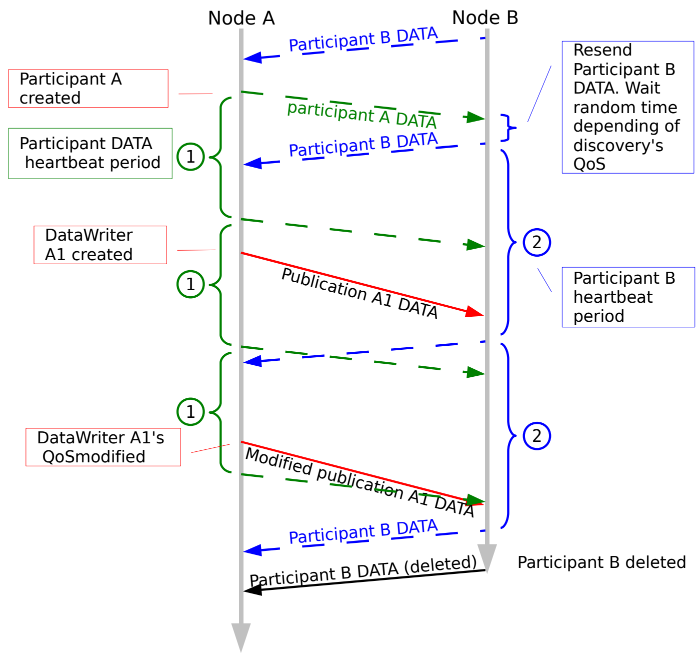


Figure 3: Basic SDP nodes dialog.

each –created or deleted– *Endpoint*, a *Participant* sends a discovery message. The *Participants* and *Endpoints* liveliness are settled by using the topic sample acknowledgment mechanism and piggybacked heartbeats as defined by the DDS-RTPS standard [25].

Fig. 3 shows a typical nodes dialog in which both SPDP heartbeats and SEDP messages are exchanged to notify *Endpoint* creation, modification and deletion.

3.2. Discovery protocol complexity evaluation. SDP analysis

The complexity evaluation of discovery procedures is a multidimensional problem for which we define a set of metrics. In broad terms, for the sake of scalability, a good discovery protocol should minimize the consumed bandwidth and the impact on the end-node resources as well. The later can be measured in terms of memory consumption and CPU usage. Memory consumption is closely related to the number of *Endpoints* the node must store. It also depends on the number of live transport-sessions (logical transport connections) that the node maintains. CPU usage is related to the amount of network traffic that the node must handle.

Table 1 defines the set of metrics to be evaluated. In our simple evaluation model, the metrics selected depend on the number of *Participants* (P) and the total number of *Endpoints* (E).

Metric name	Symbol	Meaning
$N_{participant}$	Np	Number of messages sent or received by each *Participant* if multicast is not used
N_{total}	Nt	Number of messages handled by the network if multicast is not used
$N_{mparticipant}$	Nmp	Number of messages sent or received by each *Participant* if multicast is used
N_{mtotal}	Nmt	Number of messages handled by the network if multicast is used
$M_{participant}$	Mp	Number of *Endpoints* that need to be stored on each *Participant*
$S_{participant}$	Sp	Number of live transport-sessions that need to be maintained by each *Participant*
$N_{marginalParticipant}$	Ap	Number of messages sent and received if a new empty *Participant* is added to the network
$N_{marginalEndpoint}$	Ae	Number of messages sent if a single *Endpoint* is added to one *Participant* assuming no multicast
$N_{mmarginalParticipant}$	Amp	Number of messages sent and received if a new empty *Participant* is added to the network if multicast is used
$N_{mmarginalEndpoint}$	Ame	Number of messages sent if a single *Endpoint* is added to one *Participant* if multicast is used

Table 1: DDS discovery-protocol scalability metrics.

The distribution of *Endpoints* and *Topics* within the network can influence the results expected for individual Participants. A *Domain* with most of its *Endpoints* clustered in just one *Participant* will behave differently than another with uniformly distributed *Endpoints*. Thus, to estimate the discovery traffic load the following assumptions were adopted:

- To simplify the analysis, *Endpoints* are considered to be uniformly distributed among *Participants*. In other words, the *Endpoints* per *Participant* (E/P) ratio is the same for every DDS node. Although this assumption cannot be generalized, it will not affect global resource evaluation.

- A message is only accounted as one packet. Heartbeats and ACKS are not considered.

Our study considers SDP as the reference baseline system. In SDP, each *Participant* will send its *Endpoints* information to any other *Participant*, and it will receive *Endpoints* information from every other *Participant*.

The total number of sent messages is approximately the number of *Participants* times the number of *Endpoints*. The complexity is therefore $O(P^2)$. However, if multicast is used, the number of sent messages can be reduced to the number of *Endpoints*. In any case, each *Participant* will receive a message for every *Endpoint* in the system other than its own.

Each *Participant* must store a full database containing information about any discovered *Endpoint* in the system. In a large network, most of these *Endpoints* will not be needed at all by the *Endpoints* in the given *Participant*. Therefore, a lot of extra storage is unnecessarily wasted. The discovery database is used by local *Endpoints* (present or future) to look for compatible remote *Endpoints*.

If a new *Endpoint* or *Participant* is added, a message will be sent to every other *Participant* in the DDS *Domain*. In the case of a new *Participant*, it will receive a message from every other *Participant* as well as a message announcing every existing *Endpoint*.

From now on, network traffic equations will be expressed in terms of the number of messages exchanged. Storage requirement equations will be expressed as the number of items that are requested to be stored.

The number of messages sent or received by one *Participant* is the number of messages sent and received during the SPDP for announcing the *Participant* times the number of *Endpoints* the *Participant* has to announce to any other *Participant* in the SEDP (E/P)

$$N_{Participant} = 2 \cdot (P-1) \cdot \frac{E}{P} \sim 2 \cdot E \tag{1}$$

The total number of messages sent is equal to

$$N_{total} = P \cdot (P-1) \cdot \frac{E}{P} \sim P \cdot E \tag{2}$$

If multicast is used, the number of messages can be reduced significantly. In this case, a single message can inform all network *Participants* about an *Endpoint*. Therefore, the number of messages sent or received by one *Participant* can be expressed as

$$N_{mparticipant} = E/P + (P-1) \cdot \frac{E}{P} = E \tag{3}$$

The total number of messages sent equals

$$N_{mtotal} = P \cdot \frac{E}{P} = E \tag{4}$$

The storage needed for each *Participant* is given by

$$M_{Participant} = E \tag{5}$$

The storage needed to keep the alive transport connections using the SDP is equal to

$$S_{Participant} = (P-1) \sim P \tag{6}$$

An empty *Participant* will have no *Endpoints*. If a new empty *Participant* is added to the network, the number of messages generated will be equal to

$$N_{marginalParticipant} = 2 \cdot P + E \tag{7}$$

If a new *Endpoint* is added to the network, the number of messages generated will be

$$N_{marginalEndpoint} = P + 0.5 \cdot \frac{E}{T} \tag{8}$$

where T is the number of *Topics* in the network.

And finally, using multicast, equations (7) and (8) will reduce to

$$N_{mmarginalParticipant} = 1 + P + E \tag{9}$$

$$N_{mmarginalEndpoint} = 1 + 0.5 \cdot \frac{E}{T} \tag{10}$$

4. Bloom filters

A Bloom Filter is a space-efficient data structure which compactly represents a set of elements. It supports element insertion operations but not element deletions. BFs [2] are used in multiple fields to summarize content and manage efficient membership queries. An excellent review of BF network applications can be found in [4].

A Bloom filter is a mono-dimensional array of m bits that initially are set as equal to zero. A set of k Hash functions map elements to one position in the array. The insertion operation consists in setting the array positions given by the k Hash functions at one. The membership test operation consists in hashing the key to check whether the proper positions are set to one. If any of the positions is set to zero the tested item does not belong to the set.

The possible penalty of extremely compact BF data representation is that membership queries can turn out to be false positives; false negatives are not possible, however. In other words, there is a non-zero probability that a set of array positions would be set to one even though the element is not in the set. This probability depends on the number of Hash functions (k), the size of the array (m) and the number of items represented in the filter (n). False positive probability can be approximated by [18]:

$$FP \approx \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1 - e^{-\frac{kn}{m}}\right)^{k} \tag{11}$$

To study the influence of increasing n (number of keys) in a filter for different values of k, Fig. 4 plots the theoretical false positive rate estimation respectively according to Eq. (11) for $m = 46$ bytes. In this figure the filter lengths were estimated for storing $n = 20$ keys. Our goal here is to characterize the effect of a mismatch between filter design and operational conditions. To accomplish this, the number of keys is increased without resizing the filter. Figure 4 shows that increasing the number of Hash functions (k) is not always desirable if the filter size (m) is not increased accordingly. In general, the false positive rate increases faster for a fixed-size filter if the number of Hash functions is higher. This trend can be explained because as more Hash functions are used, more vector positions are set to one when a key is added to the filter, thus increasing the probability that all vector positions associated with a specific query be equal to one.

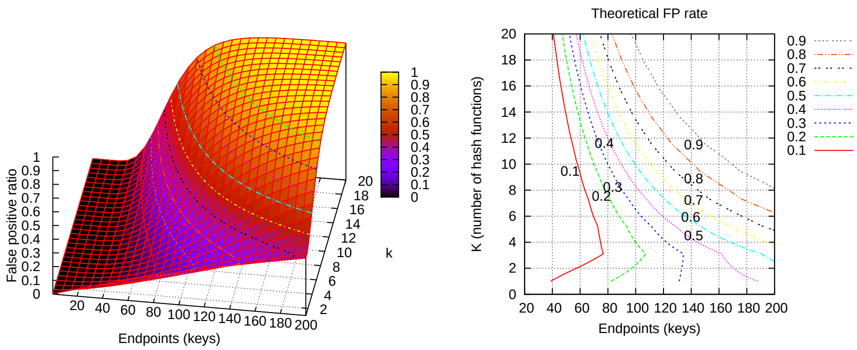


Figure 4: Theoretical BF false positive ratio for m=46 bytes.

5. SDP with Bloom filters

5.1. SDPBloom description

The Problem– Section 3.2 evaluates the SDP analytically. According to the analysis provided, for those scenarios with a high number of *Participants* and *Endpoints* per *Participant*, two problems that could be overcome are identified:

1. Memory requirements. The memory grows with the number of *Participants* and *Endpoints* since each *Participant* stores information about every entity, even those entities that are not of interest.
2. Network traffic. To distribute all the *Endpoint* information to every *Participant* a considerable traffic load is generated, especially if multicast is not available.

Our Proposal– To deal with these problems the proposal is to harness the power of Bloom Filters. The basic idea is that each *Participant* will send its own BF to other *Participants* . The filter epitomizes the *Endpoint* set in the same DDS *Domain*. Consequently, the number of messages sent to announce the *Endpoints* is reduced to a sole message containing the actual BF. Thereby, both the memory requirements and the network traffic load will be decreased. The adoption of BF changes the SDP dialog paradigm among *Participants* from "*give me all information you have*" to "*give me information to know what you have*". We call this alternative *SDPBloom*.

BFs enable each *Participant* to check if any of its *Endpoints* of interest is in the set represented by the filter. In *SDPBloom* each *Participant* stores the information about all the entities but with a significantly smaller size.

The keys or items stored in the filter must be a unique identification for *Endpoints*. We could utilize just the *Topic* name as a key to be inserted into the filter. In this case a *Participant* would make membership queries to the filter by using the *Topic* name of the local *Endpoints*. Alternatively, a more complex key can be built that is composed of the union of three elements (the *Topic* name, the *type name* and the *typecode*). The alternatives and influence of the key composition are discussed in Section 6.

SDPBloom Algorithm:

```
Require: Enabled Domain Participant
Build Bloom filter BF
while Participant is enabled do
    if Endpoint deleted then
        Rebuild BF
    end if
    for all New Endpoint E do
        Build E key Ek
        Insert Ek in BF
    end for
    Add BF to ParticipantDATA message
    for all Remote Participant filter r do
        for all Local Endpoint key Ek do
            if Ek ∈ r then
                {Try to start publication or subscription}
                Send Endpoint information (SSEDP message) to the remote Participant
                if Ek's SSEDP desired message is not received then
                    Matched Endpoint Ek is a false positive
                end if
            end if
        end for
    end for
end while
```

Figure 5: *SDPBloom* pseudocode algorithm.

Changes to SDP Participant Announcement– One relevant issue is to determine when the BF should be sent to other *Participants* in the *Domain*. As previously mentioned, the OMG DDS-RTPS standard [25] divides discovery into PDP and EDP protocols. Taking into account the purpose of each protocol, *Endpoints* information (the BF) should be included in EDP. However, –as justified in the next paragraph– we propose including the filter in the *Participant DATA* messages which are sent periodically to advertise *Domain Participants* in the *Participant* discovery procedure.

Sending the filter during the PDP has two advantages. First, this alternative is closer to the content announcement policy. Secondly, it reduces the number of messages sent to the network. More precisely, to announce its presence and its *Endpoints* filter, a *Participant* will send $P-1$ messages, where P is the number of *Participants* in the network. This means that one message is issued for each *Participant* in the *Domain*. On the other hand, sending the filter after the PDP would imply an extra message sent to the network to announce the filter, so the total messages for announcing the *Participant* and its *Endpoints* would be equal to $2 \cdot P$.

SDPBloom is described in pseudocode in Fig. 5. Additionally, Fig. 6 shows a typical sequence diagram. To highlight traffic load reduction, both SDP and *SDPBloom* network messages are shown. The filter rebuild period (if there are changes in the entities) and sent period are also represented. This last event can occur after a specified period or can be requested after a *Participant DATA* message reception.

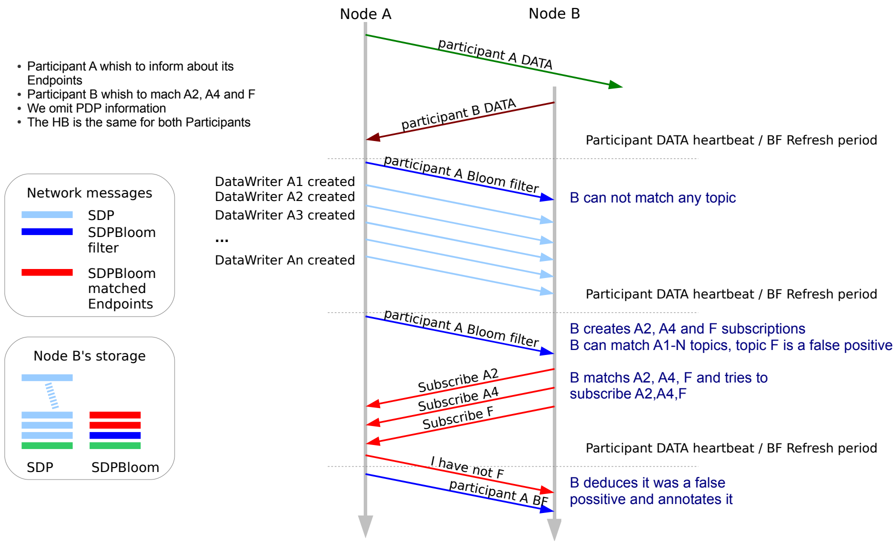


Figure 6: *SDPBloom* nodes dialog.

Given that BFs store keys the same way that Hash tables do, non-key attributes –such as QoS parameters– cannot be inserted on the filter *a priori*. The filters only allow membership queries but unfortunately they do not support range queries, such as checking a range of a QoS attribute. For example, the filter can store a key such as "Radar 02" to support matching *Topic* name operations, but it is not valid for storing QoS parameters, such as an offered deadline time. Therefore, by default a remote DDS node cannot check whether a deadline period in the filter is smaller or bigger than its accepted deadline. Our proposal is to include an *intelligent/assisted Endpoint* discovery phase. Once a *Participant* receives the remote BF, all the topics that might be of interest for this *Participant* must be checked. If the query result is positive (that is, the *Participant* is interested in a topic announced in the filter), then a modified version of SEDP protocol is used to exchange QoS settings and other parameters. If they are compatible, the *Endpoints* will be matched at the end. This approach can reduce the traffic load, given that most publication-subscription related information will be sent only in case of potential matching. Since SEDP always sends the whole Endpoints set information, we call *Selective SEDP (SSEDP)* to the slighted SEDP modification for selective Endpoints information interchange.

5.2. SDPBloom analysis

This section we analyses the *SDPBloom* algorithm. For a clearer comparison, *SDPBloom* analysis is here presented by extending the SDP study in Section 3.2.

Let us define *Matched Endpoints* (ME), as the average ratio of the number of matched *Endpoints* over the total number of *Endpoints* for each *Participant*.

After matching one or several *Endpoints*, some information (SSEDP messages) must be transferred to start the publication-subscription procedure. ME controls the number of sent and received messages during this process. In real scenarios, for instance in the naval frigate example, almost invariably it holds that $ME \ll E$.

In *SDPBloom* each *Participant* only sends and receives a filter representing each other *Participant* in the network. Therefore,

$$N_{Participant} = 2 \cdot (P-1) \cdot (ME \cdot (E/P)) \sim 2 \cdot ME \cdot E \tag{12}$$

The total number of messages sent is equal to the number of *Participants* multiplied by what each *Participant* sends, *i.e.* one BF

$$N_{total} = P \cdot (P-1) \cdot (ME \cdot (E/P)) \sim P \cdot ME \cdot E \tag{13}$$

When multicast is utilized, there is only one message for advertising a *Participant* multiplied by the *Matched Endpoints* traffic, therefore

$$N_{mparticipant} = 1 + (P-1) \cdot (ME \cdot E/P) \sim ME \cdot E \tag{14}$$

The total number of messages is given by

$$N_{mtotal} = P \cdot (ME \cdot (E/P)) = ME \cdot E \tag{15}$$

The storage needed for each *Participant* is one BF plus its *Endpoints* and its matched *Endpoints*,

$$M_{Participant} = P + \frac{E}{P} + ME \cdot E \tag{16}$$

The storage needed to keep alive the transport sessions is the same as SDP, therefore

$$S_{Participant} = (P-1) \sim P \tag{17}$$

If a new empty *Participant* is added to the network, the number of generated messages is independent of E, so

$$N_{marginalParticipant} = 2 \cdot P \tag{18}$$

If a new *Endpoint* is added to the network, we are in the same case as when SDP is used, so

$$N_{marginalEndpoint} = P + 0.5 \cdot \frac{E}{T} \tag{19}$$

Using multicast, there is one message per *Participant* to announce itself and another sent by every other *Participant* with its BF, therefore

$$N_{mmarginalParticipant} = 1 + P \sim P \tag{20}$$

$$N_{mmarginalEndpoint} = 1 + 0.5 \cdot \frac{E}{T} \tag{21}$$

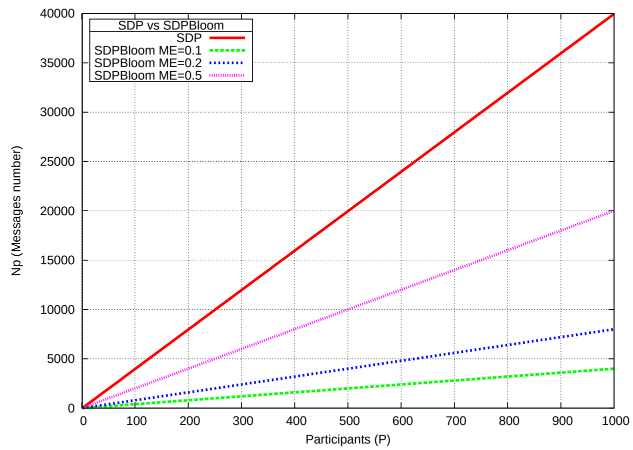


Figure 7: SDP and *SDPBloom* participant messages (metric Np).

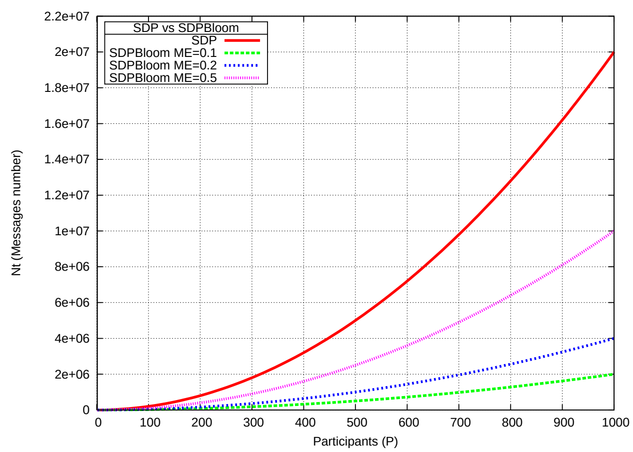


Figure 8: SDP and *SDPBloom* total network messages (metric Nt).

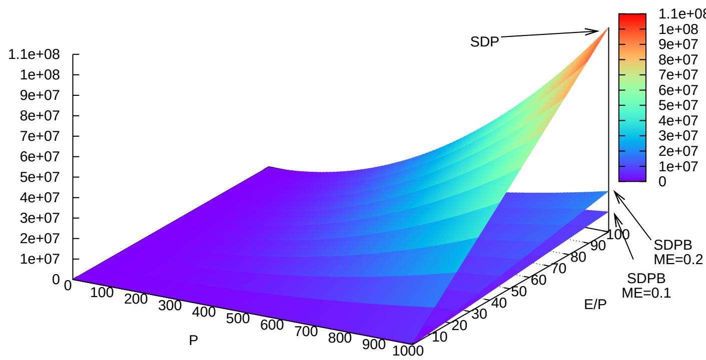


Figure 9: Total network messages (metric Nt) *versus* P and E/P.

Taking the naval frigate example again, note that *SDPBloom* halves the number of messages sent and received by each *Participant*. In addition, storage needs for tracking entities of interest (Mp) are also reduced from 2000 items (SDP) to 1120 in *SDPBloom*.

Curves depicted in Figs. 7 and 8 have been generated according to the previous SDP and *SDPBloom* analytical study. Here, performance is measured in terms of the number of network messages. For the sake of simplicity, memory consumption is not shown, although similar trends can be expected for storing remote *Endpoints* information. In the Figures we assume a uniform $E/P = 20$ ratio for all the *Participants*. So, as the number of *Participants* grows, the number of *Endpoints* increases accordingly. For *SDPBloom* curves, different points are obtained by varying the *Matched Endpoints* ratio.

After conducting the study, we conclude that *SDPBloom* can reduce system resource requirements since memory consumption and network messages are significantly smaller. However, the adoption of *SDPBloom* is conditioned by the specific network scenario. More precisely, the improvements are more significant as the number of *Endpoints* per *Participant* (E/P) increases. In addition, as the ME factor grows, *SDPBloom* approaches SDP performance. This can be explained since $ME = 1$ means that every *Participant* is interested in all the possible *Endpoints* in the network.

Additionally, Fig. 9 plots the relation between metric Nt –defined in Table 1–, the number of *Participants* P, and the E/P ratio. It can be seen that *SDPBloom* improves SDP performances for different values of E/P. In the worst case, when $E/P = 1$, *SDPBloom* performance will be similar to SDP.

6. Experimental results

To validate the analytical study in Section 5.2 and to determine the *SDP-Bloom* benefits in real scenarios, we developed a testing tool referred to as `sdpb_tester`.

6.1. Experimental framework

The `sdpb_tester` was developed in C++. It utilizes the Open Bloom Filter implementation by Arash Partow [26] and the Real-Time Innovations, Inc. DDS implementation [29]. It implements the algorithm described in Fig. 5.

It automatically creates a set of different *Topics* and *Endpoints* in a single node or a set of nodes. The `sdpb_tester` collects discovery information in the node and publishes it in the DDS Global Data Space according to SDP or SDPBloom procedure. In the case of SDPBloom, keys representing entities are stored in a BF and are included in the *SPDPdiscoveredParticipantData* messages. The `sdpb_tester` reports on the size of the discovery information for both SDP and *SDPBloom* schemes. Thereby, the `sdpb_tester` can measure the achieved compression ratio. More precisely, the `sdpb_tester` supplies the following functionalities:

- It behaves as a *SDPBloom Publisher*, or as a SDPBloom *Subscriber* or as both of them for discovery information.
- To test the data *typecode* impact on the filter key composition, it can create a set of *Endpoints* that has two different data types.
- It also allows parameterizing the following BF options: the false positive probability rate, the number of keys that will be stored in the filter *a priori* and the number of Hash functions to use. In addition, it can customize the key that will be inserted into the filter. It deals with any combination of *Topic name*, *type name* and *typecode* for the key.
- It can collect local discovery information and store it on the filter.
- It checks local *Endpoints* against the filter received.

In Open Bloom Filter implementation [26], the filter length is determined by considering two input parameters, namely the desired false positive rate and the estimated number of keys that the filter will store.

To identify the *Endpoints*, the `sdpb_tester` builds a unique key by adding a set of strings. For example, the key:

```
"WSimple Type 0Simplestruct Simple {string<255> msg}"
```

represents a key in which:

- `W` specifies that the *Endpoint* is a *DataWriter*.
- `Simple Type 0` is the *Topic* name.

- `Simple` shows the *type name*.
- `struct Simple string<255> msg;;` is the *typecode*.

A remote *DataReader* of the same *Topic* and *type* just needs to build the same key and check if it is in the filter.

6.2. Data types IDL description

The test used the *SPDPdiscoveredParticipantData* (SPDP protocol), *PublicationsBuiltinTopicData* (SEDP/SSEDP protocol) and *SubscriptionBuiltinTopicData* (SEDP/SSEDP protocol) data structures defined in the DDS-RTPS standard. Additionally, Listing 1 provides the IDL (*Interface Description Language*) [23] description of the data types used for the tests conducted. The test types examples considered are the *Simple* and *Complex* types. The *DiscoveryBloomFilter* is the extra data that have been added to the *SPDPdiscoveredParticipantData* structure to include the BF in the SDPBloom PDP announcements.

```
struct DiscoveryBloomFilter {
    sequence<octet,1024> pbf; # Bloom filter bytes vector
    short keys_number; # Number of keys
    float fp_prob; # False positive probability
};

struct Simple {
    string msg;
};

struct Complex {
    string msg;
    octet flag;
    short length;
    float temperature;
    long size;
    octet bytes_matrix[100];
};
```

Listing 1: Types IDL description.

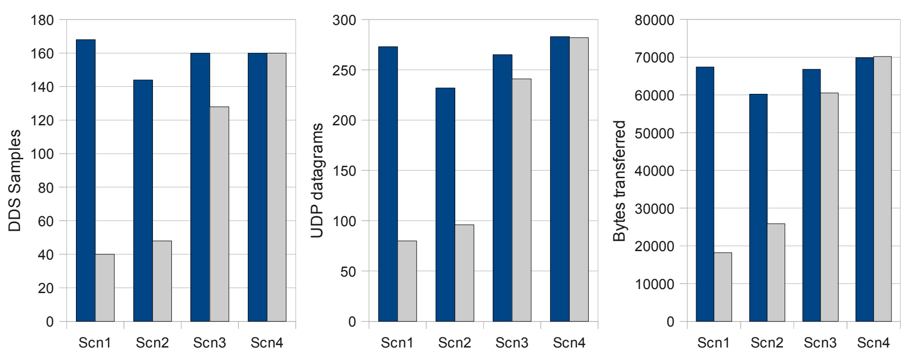
6.3. Simulation tests and results

Multiple tests have been conducted in order to analyse different aspects of the method proposed. Subsection 6.3.1 presents general network performance metrics comparing SDP and SDPBloom. Subsections 6.3.2 and 6.3.3 highlight the benefits of using BF compared to the alternative of simply modifying SDP to include the *Endpoints* list as “plain text” in SPDP announcements. Discovery data size and compression ratio are studied in terms of the different keys composition structure and different data types. Finally, Subsection 6.3.4 studies the issue of false positives.

6.3.1. DDS Samples, UDP datagrams and bandwidth

	Scenario description							
	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
	App1	App2	App1	App2	App1	App2	App1	App2
Simple-w	10	8	10	10	10	2	10	0
Simple-r	0	2	0	3	0	8	0	10
Complex-w	10	8	10	0	10	2	10	0
Complex-r	0	2	0	3	0	8	0	10
ME	0.2		0.6		0.8		1.0	
	Experimental results							
	SDP	SDPB	SDP	SDPB	SDP	SDPB	SDP	SDPB
DDS Samples	168	40	144	48	160	128	160	160
UDP Datgs.	273	80	232	96	265	241	283	282
Bytes trans.	67412	18176	60192	25856	66812	60536	69832	70172

Table 2: Example-SDP-SDPB scenario description.



(a) DDS Sampled published (b) UDP datagrams sent (c) Bytes transferred

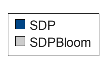


Figure 10: SDP and *SDPBloom* comparison.

The first experiments measure DDS Samples, UDP datagrams and bytes transferred to the network. DDS Samples are shown because they represent a measure that is independent of UDP related QoS parameters, which can vary the amount of final UDP datagrams sent to the network. Two applications containing a *Participant* with different DDS entities were run in an Ethernet network. All DDS entities were created using default QoS parameters and the transport was set up solely to UDPv4. The *Endpoints* key composition was the *Topic* name and *type* name. Then applications were executed during 16 seconds, publishing *SPDPdiscoveredParticipantData* discovery information every 4 seconds in order to simulate an initial discovery procedure. The Open Bloom filter needs two parameters to create the filter, the desired false positive rate (FP) and the number of keys (n) that will store the filter *a priori*. Then, the optimal number of hash functions and size of the filter are optimally estimated by the Open Bloom filter in order to fit them to the FP and n values. For all the experiments in this subsection the FP and n values were set at 0.005% and 20 respectively. Then, the bytes vector size was adjusted to 40 Bytes. This is the main extra data which is added to the *SPDPdiscoveredParticipantData* for SDPBloom. Network data were collected and analysed with *TShark*[9]. Finally, it is interesting to note that no false positives occurred during these experiments.

Table 2 describes the four scenarios under consideration. For each entity type in the table, a different DDS Topic was created so that Simple DW number one only matches a Simple DR with Topic number one. For example, scenario 1 is composed of application *App1*, which has 10 Simple DWs (associated with 10 Topics) and 10 Complex DWs (associated with 10 different Topics), and application *App2*, which is composed of 8 Simple DWs, 2 Simple DRs (linked with 2 Simple DWs at *App1*)), 8 Complex DWs and 2 Complex DRs (linked with 2 Complex DWs at *App1*), i.e. $ME = 0.2$ for the whole scenario. The scenarios differ mainly in the *Matched Endpoint* ratio (ME) in order to check the impact of this parameter on SDPBloom performance. Experimental result numbers are also shown in Table 2. The performance results of SDP and SDPBloom can be easily compared in Fig. 10. That figure reveals that SDPBloom differences with SDP are most remarkable when the ME factor decreases (see scenarios Scn1 and Scn2). The third scenario ($ME = 0.80$) represents a trade-off point where the performance of SDPBloom is closer to SDP. In this situation the decision to use SDP or SDPBloom would depend on specific environment restrictions. The fourth scenario (Scn4) represents an environment where all the Endpoints are interested in all the Endpoints ($ME = 1.00$); in this situation SDPBloom would not be suitable. It should be pointed out that most traffic is due to the SEDP (or SSEDP) protocol (*PublicationsBuiltinTopicData* and *SubscriptionBuiltinTopicData* messages). SDPBloom traffic reduction is due to the intelligent SSEDP message exchange that occurs once a remote Endpoint matching is found on the filter.

In addition, Scn3 in Figures 10b and 10c exhibits the relative differences between SDP and SDPBloom compared to the differences in Figure 10a which shows the DDS samples sent. This behavior can be explained since SDPBloom's *SPDPdiscoveredParticipantData* messages are larger than the SDP ones, so they are sliced in more UDP datagrams than SDP's *SPDPdiscoveredParticipantData*.

6.3.2. Key composition influence

The analytical study did not consider the BF key composition utilized for discovery matching (Section 5.1). However, as mentioned in Section 3.1, the *Topic* name, the *type name* and the *typecode* tuple is often sent in the discovery process. To compare SDP and *SDPBloom*, the BF key-composition influence is measured in the following experiments. This evaluation is directly related to the consumed bandwidth to announce Participant's entities if BF were not used.

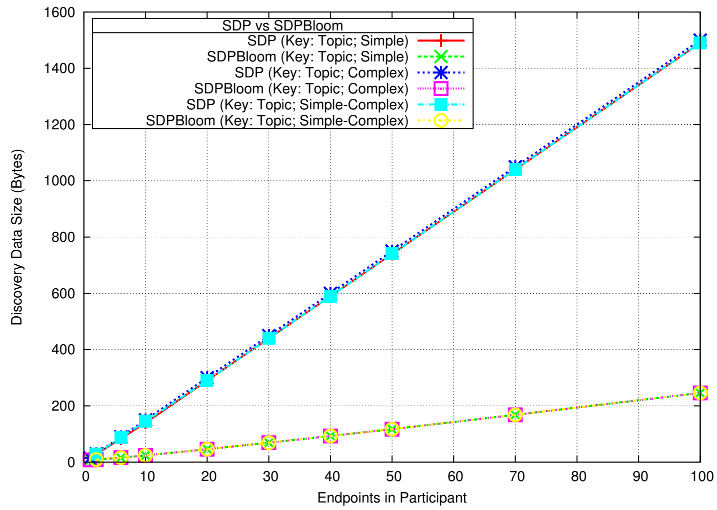


Figure 11: SDP vs *SDPBloom* (Key topic).

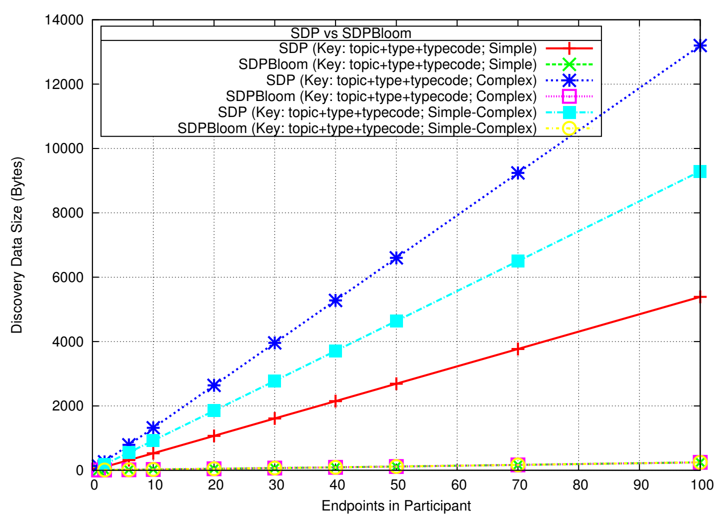


Figure 12: SDP vs *SDPBloom* (Key topic+typename+typecode).

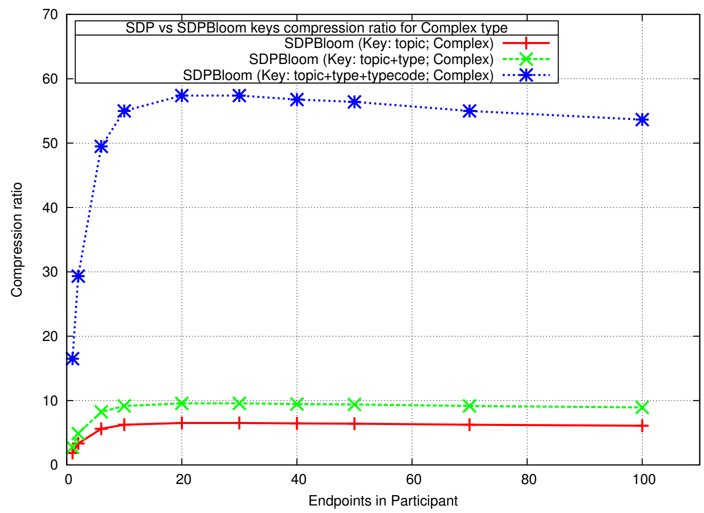


Figure 13: SDP vs *SDPBloom* compress ratio for *Complex* type.

In Figs. 11 and 12, the x axis shows the number of *Endpoints* created in a *DomainParticipant* whereas the y axis represents either the sent discovery data size or the compression ratio. Three different tests are reported: using the *Simple*, *Complex* or both data types. As can be noticed in all the evaluations reported, the reference SDP data stays constant.

Fig. 11 shows SDP and *SDPBloom* using the *Topic* name as the key for *Endpoint* matching. Since the *Topic* name is the same for the three tests, obviously the three tests provide identical results for SDP and *SDPBloom*. However, in Fig. 12 the data *type name* and *typecode* are added to the key composition. Results bear out that the best data compression is obtained with the larger *typecode*, *i.e.*the *Complex* type.

6.3.3. Compression ratio for each type and key combination

Figs. 13 and 14 plot the *SDPBloom* gain compared to the SDP baseline scheme in terms of data size compression as a function of the number of *Endpoints* per *Participant* (E/P) for different key compositions. Results for both *Complex* as well as *Simple* and *Complex* types are respectively depicted. In both cases, the best improvement is obtained when the *SDPBloom* key includes the *Topic* name, the data type and the *typecode*.

Regarding the key data type of the *Endpoints*, the *Complex* type provides a higher compression ratio (approximately up to 50:1) as was to be expected. This makes sense since *Complex* typecode is the largest, as mentioned in Section 6.3.2. More interestingly, results show that the compression ratio tends to stabilize rapidly (approximately for $E/P \geq 10$). For $E/P \leq 20$ better compression ratios are obtained as the number of *Endpoints* per *Participant* increases. As explained in Subsection 6.3.1, the Open Bloom filter tries to estimate the k and

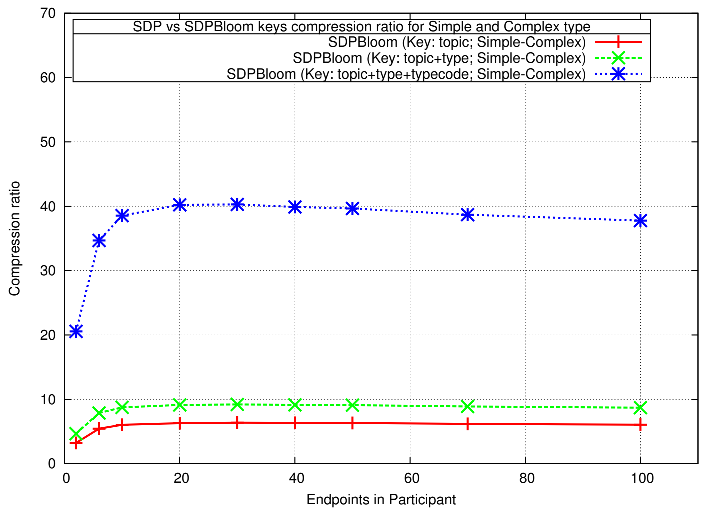


Figure 14: SDP vs *SDPBloom* compress ratio for *Simple* and *Complex* types.

the bytes vector length optimally within an iterative procedure. Therefore, the data size of the keys grows uniformly while the estimated filter size does not. This explains the decrease in the compression ratio in Figs. 13 and 14.

6.3.4. False positives

To tune the filter design, the false positive (FP) ratio should be taken into consideration. It is important to set up the filter parameters according to the envisaged use of the filter. For instance, the false positive ratio will influence in algorithm performance since the filter should be rebuilt, by using a larger array, when the estimated FP ratio exceeds a given threshold.

In particular, for all the experiments reported, a filter has been created for storing up to 20 *Endpoints* per *Participant* (keys) with an initial target-estimated false-positive ratio of 0.0015. These values yield a 46 byte vector. In the experiment conducted, a discovery data *Publisher* creates a set of *Endpoints* and adds it to the filter, while keeping the filter vector size constant. To increase the FP rate, *Endpoints* (keys) are added to the filter gradually. A discovery *Subscriber* creates a large set of false keys (outliers) and checks if the false keys are in the received filter. This can be checked in Fig. 15. For $k = 15$ if 10 extra Endpoints keys are added without resizing the filter, the FP rate is increased to 0.045, however, if $k = 5$ the FP ratio only grows up to 0.01.

Results are depicted in Fig. 15. Theoretical FP rate values are obtained according to Eq. (11). It can be pointed out that the experimental false positive rates are greater than theoretical ones. [21] shows the differences between theoretical and simulation filter error rates as well. In this respect, in [3] the authors claim that:

> Mullin [21] and Gremillion [10] both observe that the false-positive

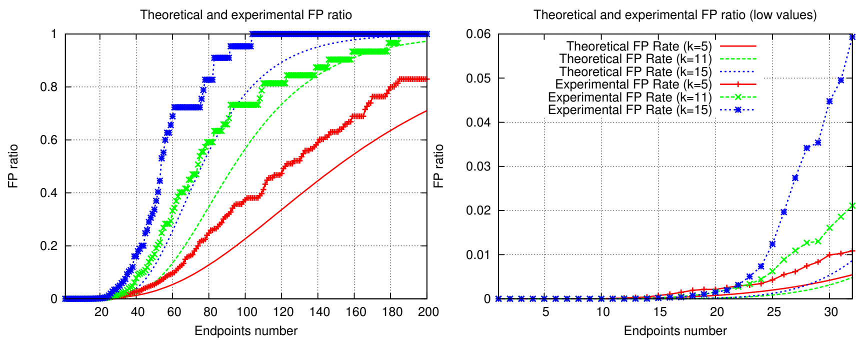


Figure 15: Experimental false positive rate (0-32 Endpoints).

> rate of Bloom filters in their database applications are slightly higher than p^k . However, they attribute this to poor quality pseudorandom numbers. Our results offer another possible explanation: the actual false-positive rate is higher than p^k , even if perfect random numbers are available.

Additionally, FP experiments also provided some insight into the Hash function number. If the number of DDS entities is constant, higher k value minimizes the FP rate for fixed length filters. Meanwhile, if DDS entities are added dynamically, lower k values minimize the appearance of false positives without resizing and rebuilding the filter.

7. Related work

In the real world, different DDS implementations adopt different discovery schemes. As previously mentioned, to provide interoperability, any DDS implementation must support DDS-RTPS (SDP), for example [27] and [29]. However, if interoperability is not requested other alternatives have been developed successfully. For instance, in [22] discovery is based on a centralized *Information Repository*. Currently implemented as a CORBA server, whenever a client requests a subscription for a topic, the Information Repository server locates the topic and notifies any existing publishers about the location of the new subscriber. Repository-based discovery reduces traffic between peers, although the whole discovery procedure relies on a single point. If the server fails the DDS system will not work at all. To mitigate this weakness, a Repository Federation [22] is suggested. In this case, different servers can collaborate to improve robustness if the original repository is no longer available.

RTI *Enterprise Discovery Service* [16] provides a similar pluggable centralized approach that also reduces steady state traffic, because *Participants* in this case only have to maintain liveliness with the server, not with every peer.

Other lightweight DDS discovery approaches have also been proposed, such as the static *Low Bandwidth Discovery Plug-in* [7]. This unidirectional scheme gets the information about remote entities (including endpoint QoS settings) from a local file. In spite of reducing bandwidth, its static nature makes this approach unsuitable for dynamic application deployments.

In comparison to previous schemes and SDP, *SDPBloom* reduces the number of messages and saves traffic load. Additionally, it is DDS-RTPS compliant (thus interoperability is satisfied), it is not static (therefore, it is suitable for dynamical deployments) and finally, it is not characterized by the single-point of failure weakness like other centralized schemes.

Up to the authors' knowledge, there are no specific papers focused on DDS discovery. Beyond the DDS context, the discovery issue has been addressed in multiple environments which include network operating systems [15], mobile communications [20], agents platforms [5], and peer-to-peer networks [19] among many others. Resource and service discovery are studied in [1] in order to summarize different technologies and to provide guidelines for selecting these alternatives for large-scale multi-domain networks.

[14] provides a discovery protocol taxonomy in which the most relevant contributions are summarized. Basically, classic approaches are based on a centralized client-server paradigm. Alternatively, other paradigms have been introduced [31]: the centralized peer-to-peer, pure peer-to-peer and hierarchical peer-to-peer[17]. DDS discovery can be classified as pure peer-to-peer.

Peer-to-peer architectures can be divided into structured and unstructured systems [19]. A structured system maintains a well-defined organization among participating nodes. Objects are placed on these nodes based on logical identifiers calculated by pre-defined functions. However in DDS, instead of organizing its nodes and entities into a structure –such as Distributed Hash Tables (DHT) implemented in Chord [32], Pastry [30] and others–, unstructured communication paths are built by the DDS *Topic virtual channel* concept. In this sense, DDS can be classified as an unstructured P2P scheme.

According to [5], in P2P architectures there are two approaches for distributing discovery information: *push* and *pull* strategies. In the *push* methodology, a node or an entity in a node sends unsolicited advertisements to other nodes. The *pull* approach explicitly sends information requests to nodes in the network. For example, Chord and Pastry schemes implement the *pull* strategy for information distribution. In those systems, nodes demand information keys from neighbor nodes that answer or forward the requests. The neighbor nodes themselves can answer, or alternatively, they can forward the request to other nodes. In general, publication-subscription architectures use the announcement approach, and more precisely, SDP adopts the advertisement methodology.

Interestingly enough, other schemes that use BFs for discovery have also successfully devised, for example, [6] includes BFs for Service Discovery Service (SDS). In particular, for mobile ad hoc networks, Liu and Heijenk [18] propose to use the attenuated BF variant for context discovery, and more recently, Yu et al. [34] for service discovery.

8. Conclusions and future work

In this paper we have provided an analytical evaluation framework for DDS discovery protocols. In particular, the Simple Discovery Protocol is evaluated as the baseline reference scheme. We propose the *SDPBloom* alternative to overcome the scalability limitations found in SDP. Our approach exploits Bloom-filter advantages for compact data representation. Along with the analytical study, we have developed a testing tool for evaluating the expected performance of our proposal in practical scenarios.

After conducting the experiments, we conclude that *SDPBloom* can improve the discovery process in DDS applications (in terms of network load and node resource consumption), especially in those scenarios with large E/P ratios and low-medium ME values. Note that even for non-large scenarios, there are distributed applications with a great number of DDS *Topics* per *Participant*. Of course, all will depend on the specific application requirements, on the particular variety of events or, on the actual managed information.

General favorable environments for *SDPBloom* have been identified. It is remarkable that, even with these favorable conditions, advantages of SDPBloom are not relevant in those scenarios where the relative amount of discovery traffic is not significant compared to the global amount of DDS traffic. Moreover, scenarios with many Endpoints with common Topics -but with heterogeneous and incompatible QoS settings- would not be benefited by our solution since SSEDP messages will be sent even when incompatible QoS setting between Endpoints exist. However, these unfavorable conditions do not imply drawbacks of using *SDPBloom* compared to SDP.

Bloom filters present a potential drawback: they are not able to list the content of the actual filter. However, DDS middleware implementations check DDS entities presence by analyzing the discovery traffic. This facility is usually carried out for debugging purposes or for providing DDS entities presence reliability. Bloom filters can introduce more difficulties into the debugging process because checking the content of a filter implies *a priori* knowledge of the items that are stored in it.

A possible solution could be to utilize the current SEDP. Thereby, the *Endpoints* discovery information could be retrieved on-demand, as occurs when there is a filter match between two *Participants*. If the process is facilitated by using network analyzing tools, the solution can be to set up the DDS system to periodically publish entities information. Hence, a network analyzing program could read the *Endpoint* information traffic. In this case, the publishing frequency does not necessarily need to be the same as the *Participant* announcement frequency and it could be controlled by the adoption of a given DDS QoS policy.

Interoperability of the proposed discovery protocol with DDS instances not equipped with SDPBloom must be studied. In DDS-RTPS document Section 8.5.1, it is said that "Implementations may choose to support multiple PDPs and EDPs", however, it is not established how the discovery protocol must be selected from a set of available protocols. To inter-operate any pair of participants one possible approach for selecting the particular protocol could be as follows: each Participant could announce the list of eligible discovery protocols (depending on the DDS implementation) ranked according the application designer preferences. In this way they could agree to use the most preferable common protocol. Additionally, to assure interoperability -according to the standard- SPDP and SEDP must always be present in the announced list, typically with the lowest rank. These details -very coupled to the particular implementation- deserve a deeper study in future works.

Summarizing, the advantages of adopting the proposed *SDPBloom* approach in DDS are:

- The number of messages sent to the network for *Endpoint* advertisement in *SDPBloom* stays constant while the number of *Endpoints* increases.
- The Participant's *Endpoints* information exchange is reduced to the information of the matched *Endpoints*, which is significantly smaller than the total *Endpoints* number of SDP.
- The improvements are better in scenarios with a high number of *Endpoints* per *Participant*.
- The more information that is added to the key, the better the compression ratio provided. In this sense, it is noteworthy how the *typecode* especially increases the compression ratio.

In addition to the previously identified *SDPBloom* advantages, the following *SDPBloom's* drawbacks were found:

- Given the extremely compact information representation, the debugging process can be more difficult. However, as previously mentioned, there are some feasible solutions to alleviate this problem.
- In *SDPBloom*,the false-positive *Endpoints* matching probability is greater than zero. This non-desirable behavior could make the algorithm non-deterministic. For critical real-time applications this might not be no acceptable; therefore, as future work this issue demands further study.

Finally, it should be pointed out that false positives probability is an increasing function of the key length. Related to that, there is always an optimal trade-off between achieved bandwidth reduction versus the CPU increase because of the use of BF. [28] propose several new BF variants for flexible trade-off between false positive rate, space efficiency, cache-efficiency, hash-efficiency, and computational effort. Deeper study of these issues and its potential adoption in *SDPBloom* remain for future works.

Acknowledgments

We explicitly thank the anonymous reviewers, whose valuable comments definitively helped to improve the quality of this manuscript. The research of

Javier Sánchez-Monedero has been funded by the *Junta de Andalucía* Ph. D. Student Program. This research was subsidized in part by the *Plan Propio de la Universidad de Granada, 2010* and by the *Ministerio de Ciencia e Innovación* of the Spanish Government (project TIN2009-13992-C02-02).

References

[1] R. Ahmed, N. Limam, J. Xiao, Y. Iraqi, R. Boutaba, Resource and service discovery in large-scale multi-domain networks, Communications Surveys Tutorials, IEEE 9 (2007) 2–30.

[2] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM 13 (1970) 422–6.

[3] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, Y. Tang, On the false-positive rate of bloom filters, Information Processing Letters 108 (2008) 210 –3.

[4] A. Broder, M. Mitzenmacher, Network Applications of Bloom Filters: A Survey, Internet Mathematics 1 (2005) 485–509.

[5] C. Campo, C. García-Rubio, A.M. López, F. Almenárez, PDP: A lightweight discovery protocol for local-scope interactions in wireless ad hoc networks, Computer Networks 50 (2006) 3264 –83.

[6] S.E. Czerwinski, B.Y. Zhao, T.D. Hodes, A.D. Joseph, R.H. Katz, An architecture for a secure service discovery service, in: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, MobiCom '99, ACM, New York, NY, USA, 1999, pp. 24–35.

[7] eProsima, Low Bandwidth Discovery Plug-in. http://www.eprosima.com/directDownloads/DiscoveryPlugin.pdf, .

[8] U. Forum, UPnP white paper, http://upnp.org/sdcps-and-certification/resources/whitepapers/, 2006.

[9] W. Foundation, Wireshark.http://www.wireshark.org/, 2010.

[10] L.L. Gremillion, Designing a bloom filter for differential file access, Communications of the ACM 25 (1982) 600–4.

[11] E. Guttman, C. Perkins, J. Veizades, M. Day, Service Location Protocol, Version 2, Standards Track. RFC 2608, Internet Engineering Task Force., 2002.

[12] H. van 't Hag, DDS Scalability: One size fits all?, in: Real-time and Embedded Systems Workshop, Arlington VA, USA.

[13] S. Helal, N. Desai, V. Verma, C. Lee, Konark - a service discovery and delivery protocol for ad-hoc networks, in: IEEE Wireless Communications and Networking Conference (WCNC 2003), volume 3, pp. 2107 –13.

[14] J. Hoffert, S. Jiang, D.C. Schmidt, A taxonomy of discovery services and gap analysis for ultra-large scale systems, in: ACM-SE 45: Proceedings of the 45th annual southeast regional conference, ACM, New York, NY, USA, 2007, pp. 355–61.

[15] T.A. Howes, M.C. Smith, G.S. Good, Understanding and Deploying LDAP Directory Services, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[16] R.T.I. Inc., RTI Data Distribution Service. Users' Manual (version 4.5), 2010.

[17] N. Limam, J. Ziembicki, R. Ahmed, Y. Iraqi, D.T. Li, R. Boutaba, F. Cuervo, OSDA: Open service discovery architecture for efficient cross-domain service provisioning, Computer Communications 30 (2007) 546–63.

[18] F. Liu, G.J. Heijenk, Context discovery using attenuated bloom filters in ad-hoc networks, Journal of Internet Engineering 1 (2007) 49–58.

[19] E. Meshkova, J. Riihijrvi, M. Petrova, P. Mhnen, A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks, Computer Networks 52 (2008) 2097 –128.

[20] A.N. Mian, R. Baldoni, R. Beraldi, A survey of service discovery protocols in multihop mobile ad hoc networks, IEEE Pervasive Computing 8 (2009) 66–74.

[21] J.K. Mullin, A second look at bloom filters, Communications of the ACM 26 (1983) 570–1.

[22] Object Computing, Inc., OpenDDS. http://www.opendds.org, 2010.

[23] Object Management Group, OMG, OMG IDL Syntax and Semantics - Common Object Request Broker Architecture (CORBA), v3.0, 2002.

[24] Object Management Group, OMG, Data Distribution Service for Real-time Systems specification, version 1.2, 2007.

[25] Object Management Group, OMG, The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol specification, version 2.1, 2009.

[26] A. Partow, General Purpose Hash Function Algorithms. http://www.partow.net/programming/hashfunctions/index.html, 2009.

[27] PrismTech Ltd., OpenSplice DDS. http://www.opensplice.com, 2010.

[28] F. Putze, P. Sanders, J. Singler, Cache-, hash- and space-efficient bloom filters, Experimental Algorithms, 6th International Workshop, WEA 2007, Springer-Verlag, Lecture Notes in Computer Science 4525 (2007) 108–21.

[29] Real-Time Innovations Inc., RTI Data Distribution Service. http://www.rti.com, 2010.

[30] A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, in: Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Springer-Verlag, 2001, pp. 329–50.

[31] R. Schollmeier, A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications, in: First International Conference on Peer-to-Peer Computing, pp. 101–2.

[32] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Transaction on Networking 11 (2003) 17–32.

[33] J. Waldo, K. Arnold, The JINI specification., Addison-Wesley, Reading, MA, 2nd edition, 2000.

[34] Y. Yu, Y. Zhou, S. Du, Service discovery in mobile ad hoc networks using mobility-aware attenuated bloom filters, in: Services Science, Management and Engineering, 2009. SSME '09. IITA International Conference on, pp. 266 –9.