MASTER THESIS

A DDS Discovery Protocol based on Bloom filters

Javier Sánchez Monedero
Director: Juan Manuel López Soler

Granada, 14th September 2009
A DDS Discovery Protocol based on Bloom filters

Javier Sánchez Monedero

KEYWORDS: DDS, Data Distribution Service, discovery, bloom filters, middleware.

Abstract

The DDS middleware (Data Distribution Service) has been used for addressing the data distribution and interoperability between different networks, architectures, and operating systems.

A crucial point in the integration of these heterogeneous environments relies on the discovery process. A DDS discovery protocol matches publisher and subscriber entities (Endpoints) placed in different network nodes. The scalability of a DDS infrastructure is limited by the discovery protocol, hence the discovery process is one of the most important features of a DDS implementation.

This master project addresses the design and evaluation of alternatives to the current DDS discovery protocol, the Simple Discovery Protocol (SDP). These alternatives should improve the scalability of the SDP protocol for allowing the use of DDS in larger scenarios.

In SDP, the number of sent messages and the consumed network bandwidth grow with the number of Endpoints. Additionally, each node consumes memory for storing all the remote node’s discovery information. Basically, the total number of messages sent over the network is $P \times E$ where $P$ is the number of Participants and $E$ the number of Endpoints with $E \gg P$.

We propose to improve the DDS discovery process by using a compact information representation technique called Bloom filter. We propose to modify the SDP for using Bloom filters. We call this proposal SDPBloom. By using Bloom filters the number of messages sent over the network is closer to $P \times P$ instead of $P \times E$. Additionally, both the spent network bandwidth consumption and nodes memory usage decrease. The analytical results show the improvement of SDPBloom over SDP. In addition, we have implemented a test application for measuring the real advantages of using Bloom filters for representing discovery information. The analytical and experimental results show that the number of messages sent to the network for announcing Endpoints is still constant with no dependency with the Endpoints number ($E$). As it will be shown, under some circumstances the proposed approach can provide a discovery information compression ratio equal to $50 : 1$. 
# CONTENTS

1 Introduction 1
   1.1 Introduction to middleware and discovery issues 1
   1.2 The Data Distribution Service 3

2 Simple Discovery Protocol 7
   2.1 Simple Discovery Protocol description 7
   2.2 SDP analytical study and characterization 9
      2.2.1 Scalability metrics and comparison methodology for discovery 9
      2.2.2 Simple Discovery Protocol Scalability Analysis 12
      2.2.3 Scalability equations 12

3 Discovery technologies review 15
   3.1 Functional relationship between the network entities 15
   3.2 Network topology 16
   3.3 Database and information representation 17
   3.4 Distributed Hash Tables, DHT 17
   3.5 Replication methods 19
      3.5.1 Complete information replication in servers 19
      3.5.2 Partial information replication in servers 19
   3.6 Learning, cache and distinguished local and global discovery 20
   3.7 Distribution of entities and resources announcements 21
   3.8 Comparison table in discovery 21
   3.9 Performance and scalability 21
   3.10 Related work review summary 22

4 Bloom filters 25
   4.1 Introduction and original Bloom filters 25
   4.2 Counting Bloom filters 27
   4.3 Generalized Bloom Filter 28
1.1 DDS entities relationship .................................................. 5
2.1 DCPS Builitin entities for Discovery Purposes (Source RTI DDS User’s Manual [49]) ............................... 8
2.2 DCPS Builitin entities for Discovery Purposes (Source RTI DDS User’s Manual [49]) ............................... 10
4.1 Bloom filters operations description ................................. 26
5.1 SDP Bloom algorithm description ........................................ 39
5.2 SDP Bloom nodes dialog ...................................................... 40
5.3 SDP Bloom analytical results ............................................... 44
6.1 sdb_tester command line options .................................... 46
6.2 SDP vs SDP Bloom key topic ............................................. 49
6.3 SDP vs SDP Bloom key topic+typename ............................ 50
6.4 SDP vs SDP Bloom key topic+typename+typecode ............... 50
7.1 DDS multi-site distributed application integrated with the DDS Router ................................................. 58
CHAPTER 1

Introduction

1.1 Introduction to middleware and discovery issues

During the last years computer networks have been improving quality, performance and features bringing along new paradigms for application development, specially for distributed applications. Distributed applications run simultaneously on different machines and typically are connected by networks, shared memory or file systems. These applications allow to develop new solutions which noticeably improve processing capacity and which connect large number of information producers and consumers. Distributed applications needs to match information producers and consumers during a process called discovery. Discovery includes matching information sources, resource location and so on depending of the problem environment. Discovery has received substantial interest from both researchers and industry because it is an essential point in a distributed system scalability and performance.

Distributed applications examples are the email, instant messaging, air traffic control, complex scientific simulations and more. The underlying technology will depend on the final application purpose. For instance, the requirements for an instant messaging service differ from a traffic control application. In this way a set of technologies has been proposed for supporting different environment requirements. These technologies provide features for addressing common distributed application issues. Typically these technologies are called middleware. A middleware releases the developer from common distributed applications tasks so the developer can focus on the distributed problem rather than in those common tasks.

This work focuses on Message Oriented Middleware (MOM) which is an infrastructure focused on sending and receiving messages. This sort of middleware is intended to efficiently distributing data among information producers and consumers, increasing the
interoperability, portability, and flexibility of a distributed application components.

Examples of MOM technologies are the Web Services [1] [2], the Java Message Service (JMS) [3], the Common Object Request Broker Architecture (CORBA) [4], the Message Passing Interface (MPI) [5] or the Data Distribution Service (DDS) [6].

Many real-time applications use a middleware for addressing communication between components of the distributed application. For those applications not only the data distribution feature is relevant, but also the way this transmission is done in regards to efficiency, determinism, security, reliability and error notification and processing. In this area many solutions have being designed for satisfying these requirements, and this is the case of the DDS technology.

DDS is the first international open standard for addressing publish/subscribe communications for real-time and embedded systems.

DDS has being used in relatively small networks composed of tens or hundreds on nodes running distributed applications. As the distributed systems grown in terms of network complexity, network nodes and DDS entities running on these nodes, a set of open issues appears. The using of the transport layer, the security management [7], the global monitoring and interaction interfaces, the system specification and deployment [8], the system scalability [9], the discovery process and so on are new challenges for the DDS systems.

DDS deals with integration and interoperability of heterogeneous systems by building a global data-centric space for efficient data distribution. A essential point for building the global data space is the discovery process between the publisher and subscriber entities running on different nodes. The scalability of a DDS infrastructure is limited by the discovery protocol, hence the discovery process is one of the most important features of a DDS implementation. The current discovery protocol described in the standard document The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification [10], formerly known as RTPS, is the Simple Discovery Protocol (SDP). SDP works properly in relatively small networks in terms of nodes and DDS entities running on those nodes.

The current research project deals with the analysis, evaluation and design of improvements for the current discovery protocol.

Most DDS implementations currently support only the SDP. This first approach lacks of scalability due to every Participant in a given Domain needs to know the complete list of Endpoints in each other Participant in the Domain, even for those Participant Endpoints which are not of interest. The problem stems from each node needs to know where the set of Endpoints that match to its Endpoints are located. Discovery solutions must preserve the “anonymous” publication-subscription model and the peer-to-peer architecture.

Broadly speaking, the goal is to reduce network traffic and memory requirements in nodes for the discovery protocol. Inspired by its traditional use -in data base query- and, more recently in some network applications, we propose to include Bloom filters in SDP
1.2 The Data Distribution Service

This section aims to briefly introduce the Data Distribution Service for Real-Time Systems (DDS). For extra information [12] and [13] are recommended.

The DDS standard is maintained by the Object Management Group [14]. The specification is divided into two documents:

- Data Distribution Service for Real-time Systems specification [10].

The OMG Data Distribution Service (DDS) specification provides a Data-Centric Publish-Subscribe (DCPS) communication standard for a range of distributed real-time and em-
1. Introduction

bedded computing environments, from small networked embedded systems up to large-scale information backbones.

The DCPS conceptual model is based on the abstraction of a “Global Data Space” (GDS). Publishers (applications) posts new data into this GDS, the DDS DCPS middleware propagates the information to all interested Subscribers (applications), and Subscribers read data.

For easy understanding of this project and the DDS architecture, the following concepts are explained:

**Domain**: A Domain is a virtual network concept. The DDS applications are able to send and receive data within a domain. Only the Publishers and the Subscribers within the same domain can communicate. Domains help to isolate and optimize communication within a community that shares common interests.

**DomainParticipant**: A Domain Participant (or simply Participant) represents the participation of the application on a communication plane, a Domain, that isolates applications running on the same set of physical computers from each other. It acts as an entry-point of service and a container for the other entity objects.

**Topic**: A Topic identified by its unique name in the whole domain defines the type of the data object that can be written and read on the GDS. Communication happens only if the topic published by a DataWriter matches a topic subscribed to by a DataReader. Communication via topics is anonymous and transparent. For example, Publishers and Subscribers need not be concerned with either how topics are created or who is writing/reading them.

**Publisher**: A Publisher is an object responsible for data distribution according to the Publisher’s QoS (Quality of Service). It may publish data objects of different types.

**Subscriber**: A Subscriber is an object responsible for receiving published data and making it available to the receiving application according to the Subscriber’s QoS. Subscriber reads topics in the GDS for which a matching subscription exists and informs DataReaders that the data is received.

**DataWriter**: Applications use DataWriters to write data to the GDS of a domain through Publisher.

**DataReader**: DataReaders notify an application that data are available.

Figure 1.1 shows the relationship between the explained DDS entities.

The basic model of the DDS is a unidirectional data communication in a fixed network where the Publisher pushes the relevant data into the HistoryCache of Subscribers. The DDS is based on an overlay structure where a Publisher of a topic is linked with all Subscribers for the same topic.
1.2. The Data Distribution Service

To establish communication between Publishers and Subscribers, the DDS relies on the use of a built-in discovery protocol that allows a Publisher to dynamically discover the matched Subscribers and vice-versa. The implementation of the DDS needs to provide a discovery protocol to identify the presence or absence of the other Endpoints when they either join into or leave from the network.

The discovery protocol of the DDS could lead to a possible the transparent plug-and-play dissemination of all the information between publisher and subscriber in various heterogeneous networks where the DDS middleware has been deployed. One of the key distinguishing features of the DDS when compared to other publish/subscribe middleware is its extremely rich QoS support.

The aspects of the behaviour of the DDS service rely on the use of QoS, a set of characteristics that limits the use of resources such as network bandwidth and memory, and controls many non functional properties of the topics such as persistence, reliability, timeliness, etc [12].

Figure 1.1: DDS entities relationship
1. Introduction
This chapter establishes a framework for evaluating DDS discovery protocols. To do this, an analytical study is provided for characterizing the current standard discovery protocol.

The DDS Interoperability Protocol [10] specifies that every DDS discovery protocol must be divided in two consecutive processes: the Participant Discovery Protocol (PDP) and the Endpoint Discovery Protocol (EDP). The PDP purpose is to discovery new Participants in the same Domain in a network. When a new Participant is discovered, the EDP process is started for exchanging local and remote Endpoints information between two Participants.

### 2.1 Simple Discovery Protocol description

The DDS standard specifies the Simple Discovery Protocol (SDP). It is divided in the Simple Participant Discovery Protocol (SPDP) and the Endpoint Discovery Protocol (SEDP). Every DDS and RTPS compliant implementation must support at least the SDP.

SDP uses DDS publication itself for discovery purposes. A special type of DDS entities called build-in entities are created for each DomainParticipant with predefined QoS policies. The discovery process can be tuned with specific Quality of Services policies which can be applied to these built-in entities. The SDP uses a special set of Topics, DataReaders and DataWriters for advertising and discovering Participants and Endpoints through the network. Figure 2.1 shows the topics related to SPDP (“DCPSParticipant”), and SEDP (“DCPSSubscription”, “DCPSPublication” and “DCPSTopic”). For each one of these topics there is a specific data type, for example the SDPdiscoveredParticipantData is the data type used in the “DCPSParticipant” topic.
2. Simple Discovery Protocol

The discovery process is started from a list of known hosts, this is, a list of IP addresses a Participant will announce its presence. Typically, if there are no IP addresses specified, default unicast or multicast addresses are used to announce the Participant presence in a network. Both options can be used together.

When a Participant in a node is enabled, the first discovery stage consists on to discover other Participants. This is done via the PDP and it is restricted to discover participants in the same DDS Domain. In SPDP, a special message called \texttt{SPDPdiscoveredParticipantData} or simply \texttt{Participant DATA}, is periodically sent to known peers when a DomainParticipant is created or deleted. By default, when new \texttt{SPDPdiscoveredParticipantData} messages are received the own \texttt{SPDPdiscoveredParticipantData} are sent to the remote Participant. This remote Participant is added to the internal database.

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

Similarly, the SEDP publication and subscription information is composed by data needed for matching local with remote Endpoints. These data are:

1. The Topic name of the Endpoint.
2. The data type name.
3. The data typecode, this is the data type structure description.
4. The QoS parameters such as accepted deadline, reliability level, etc.
5. Other data specified in the standard.

The DDS middleware must check that all the three first data set are the same and that the QoS parameters are consistent. If this is right, the remote Endpoint is suitable for starting the publication-subscription communication. Even though the typecode is not included in the standard discovery information, the data type description is usually included by DDS implementations for properly serializing and deserializing data and for error checking. 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 other to preserve data correctness.

In addition to the standard information for representing these entities, DDS implementation products use to include some parameters for offering extra QoS features not included in the standard.

The information of discovered participants, and associated publications and subscriptions, is stored in a local database in each Participant. The SEDP protocol sends the Endpoints information for every Participant in the local Participant database and it receives every discovered Participant’s Endpoints information. For each Endpoint a Participant sends a discovery message. The liveliness of Participants and Endpoints is determined by using ACK-NACK mechanism and a piggybacked heartbeat. Figure 2.2 shows the nodes dialog combining both SPDP heartbeat and SEDP traffic for notifying Endpoints creation, modification and deletion.

2.2 SDP analytical study and characterization

2.2.1 Scalability metrics and comparison methodology for discovery

The purpose of performing an analytical study of a discovery protocol is to perform an implementation independent evaluation of the complexity of the protocol. Also, to develop a discovery protocol is a complex task where a previous protocol design and evaluation is necessary.
2. Simple Discovery Protocol

Figure 2.2: DCPS Builtin entities for Discovery Purposes (Source RTI DDS User’s Manual [49])
In order to compare different discovery protocols it is important to develop the proper metrics. In broad terms a good scalability protocol aims to minimize the bandwidth consumed on the network as well as the impact on each individual node. The impact on a node can be measured in terms of the memory consumption and the CPU usage. The memory consumption is related to the number of endpoints the node must store as well as to the number of live transport-sessions (logical transport connections) that must be maintained by the node. The CPU usage is connected with the amount of network traffic that the node must handle.

For measuring the performance of a discovery protocol we have identified a set of metrics. These metrics are listed in Table 2.1. Those metrics depends on the number of Participants \(P\), the total number of Endpoints \(E\) and the average Endpoints per Participant \(E/P\).

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N_{\text{participant}})</td>
<td>(N_p)</td>
<td>Number of messages sent or received by each Participant if multicast is not used</td>
</tr>
<tr>
<td>(N_{\text{total}})</td>
<td>(N_t)</td>
<td>Number of messages handled by the network if multicast is not used</td>
</tr>
<tr>
<td>(N_{\text{Mparticipant}})</td>
<td>(N_{mp})</td>
<td>Number of messages sent or received by each Participant if multicast is used</td>
</tr>
<tr>
<td>(N_{M\text{total}})</td>
<td>(N_{mt})</td>
<td>Number of messages handled by the network if multicast is used</td>
</tr>
<tr>
<td>(M_{\text{participant}})</td>
<td>(M_p)</td>
<td>Number of endpoints that need to be stored on each participant</td>
</tr>
<tr>
<td>(S_{\text{participant}})</td>
<td>(S_p)</td>
<td>Number of live transport-sessions that need to be maintained by each Participant</td>
</tr>
<tr>
<td>(N_{\text{marginalParticipant}})</td>
<td>(A_p)</td>
<td>Number of messages sent &amp; received if a new empty Participant is added to the network</td>
</tr>
<tr>
<td>(N_{\text{marginalEndpoint}})</td>
<td>(A_e)</td>
<td>Number of messages sent if a single endpoint is added to one Participant assuming no multicast</td>
</tr>
</tbody>
</table>

Table 2.1: DDS discovery protocols scalability metrics

The Table 2.1 does not capture all the relevant dimensions. For example whether a protocol allows a sender to reuse the same message and send it to multiple destinations as opposed to having to build a separate message for each destination. Clearly characterizing scalability is not simple, but the metrics described above are sufficient for evaluation and comparison purposes.

The metrics listed in Table 2.1 will be used to compare the different discovery protocols. All of the metrics depend on the number of entities in the network, specifically the
2. Simple Discovery Protocol

number of Participants, the number of Endpoints and the number of Topics. The distribution of Endpoints and Topics within the network can affect the metrics; a network with the most of Endpoints clustered in one Participant will have a different behaviour that one with equally distributed Endpoints. The number of messages sent also depends on the protocol used to deliver a message. For instance, a reliable protocol will add a certain number of heartbeats and ACKS to a message. In this document, scalability analysis is performed by comparing metrics versus the network size measured by the number of Participants, Topics, and Endpoints. To reduce the number of variables, network size can be measured just by the number of Participants, assuming a fixed number of Endpoints per Participant and Endpoints per Topic. The goal is to find an estimation of the discovery traffic, to this end the following assumptions have been made:

- Endpoints are uniformly distributed among Participants.
- A message is only counted as one packet. Heartbeats and ACKS used to send the message reliably are not considered.

2.2.2 Simple Discovery Protocol Scalability Analysis

The current standard discovery protocol, the SDP, is used as the baseline to compare the proposed discovery methods. In SDP, each Participant will send its Endpoint information to every other Participant, and it will receive Endpoint information from every other Participant.

The total number of sent messages is approximately the number of Participants times the number of Endpoints, and will grow as the $O(P^2)$. However, if multicast is used, the number of sent messages can be reduced up to the number of Endpoints. Whether or not multicast is used, each Participant will receive a message for every Endpoint in the system other than its own.

Each Participant must store a full database of all of the Endpoints in the system. In a large network, most of these Endpoints will not be needed by the Endpoints in the given Participant, so a lot of extra storage is used unnecessarily. The Participant must store all of the Endpoints in order to check for a match if a new Endpoint is added to that Participant.

If a new Endpoint or Participant is added, a message will go out to every other Participant in the system. In the case of a new Participant, the new Participant will receive a message for every existing Endpoint and a message for every existing Endpoint.

2.2.3 Scalability equations

From now on, network traffic equations will be expressed in number of interchanged messages while storage requirement equations will be expressed in number of items that are requested to be stored.
2.2. SDP analytical study and characterization

In the Simple Discovery Protocol, each Participant communicates directly with every other Participant. Therefore 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 multiplied by the number of Endpoints the Participant has to inform to any other Participant in the SEDP (\(E/P\)):

\[ N_{\text{Participant}} = 2 \times (P - 1) \times E/P \sim 2 \times E \] (2.1)

The total number of sent messages equals to:

\[ N_{\text{total}} = P \times (P - 1) \times E/P \sim P \times E \] (2.2)

If multicast is used the number of messages can be reduced significantly. Using multicast, a single message can inform about an Endpoint to all the network Participants so the number of sent or received messages by one Participant is:

\[ N_{M\text{Participant}} = E/P + (P - 1) \times E/P = E \] (2.3)

The total number of sent messages equals to:

\[ N_{M\text{total}} = P \times E/P = E \] (2.4)

The storage needed for each Participant is:

\[ M_{\text{Participant}} = E \] (2.5)

The storage needed to keep the alive transport connections using the Simple protocol is:

\[ S_{\text{Participant}} = (P - 1) \sim P \] (2.6)

If a new empty Participant is added to the network, the number of generated messages is:

\[ N_{\text{marginalParticipant}} = 2 \times P + E \] (2.7)

If a new Endpoint is added to the network, the number of generated messages is:

\[ N_{\text{marginalEndpoint}} = P + 0.5 \times E/T \] (2.8)

Using multicast equations (2.7) and (2.8) are reduced to:

\[ N_{M\text{marginalParticipant}} = 1 + P + E \] (2.9)

\[ N_{M\text{marginalEndpoint}} = 1 + 0.5 \times E/T \] (2.10)

And finally, if a new Endpoint is added to the network, the number of involved nodes (Participant) affected is equal to:

\[ A_{\text{marginalEndpoint}} = P \] (2.11)
2. Simple Discovery Protocol
This chapter presents the main features of some of the most significant discovery protocols. The purpose is to identify features and alternatives in discovery to envisage solutions for improving the discovery process in DDS.

We have mainly considered the following works:

- **Chord**: a scalable peer-to-peer lookup protocol for internet applications [15].
- **Pastry**: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems [16].
- **Directory Facilitator and Service Discovery Agent** [19], we will name it DFSDA.
- **Kademlia**: A Peer-to-Peer Information System Based on the XOR Metric [20].
- **An advertisement-based peer-to-peer search algorithm** [21], also named as ASAP.
- P2P networks like eDonkey 2000 (ed2k) [22][23] or Gnutella [24][1].
- **LDAP (Lightweight Directory Access Protocol)** directories based systems [25][26][27].

### 3.1 Functional relationship between the network entities

We identify the following types of functional relations in discovery techniques:

1Since original eDonkey 2000 and Gnutella Web sites do not longer exists, Wikipedia is used as the reference source of information.
3. Discovery technologies review

- **Client-Server.** A classical centralized Client-Server application.

- **Centralized peer-to-peer.** The discovery is doing using one or several central servers. The other transmissions are done by peer-to-peer connections. This implies to define two or more types of nodes. Although the discovery nodes, the central servers, are essential, in some implementations the peers are allowed to interchange discovery information between neighbour peers.

- **Pure peer-to-peer.** All the nodes have the same responsibilities in all operations, included discovery communications. The best examples of pure P2P are Chord, Pastry, Kademlia, which implement several advanced techniques for maintaining the structure.

- **Hierarchical P2P.** In this approach several types of nodes are defined, and some levels of hierarchy are established. At least one upper level is disposed with some nodes that maintain global discovery information using pure P2P among them. This is the case of DFSDA and OSDA approaches.

3.2 Network topology

The way in which information and nodes are organized imply to define a structure, such as a ring, a tree or hierarchical topology network. The defined topology is closely related to the functional relationship.

Pastry and Chord organize its nodes into a virtual ring that is maintained by the self nodes. All nodes have the same responsibilities in the network.

INS [28] expresses service descriptions as a tree-like hierarchy of descriptive attributes and values. Discovery is performed using an application-level overlay of directory entities. Advertisements are disseminated using a logical spanning tree topology, and replicated in each directory. Although the replication strategy handles fault-tolerance well, INS does not scale to large scale networks.

LDAP directories organize information in trees which can be stored in one or multiple servers.

Solutions like OSDA combines several structures. It uses Chord’s logical ring to maintain a global directory of discovery information and a hierarchy of nodes, which are grouped and represented at global level by another kind of node. This scheme allows the differentiation of local and global discovery techniques.

Similarly, DFSDA uses an hierarchical topology as well to divide the functionality between nodes and discovery protocols. It groups nodes to allow local discovery.

Alternatives such as the ed2k network does not have a structure. The eDonkey 2000 just uses a list of servers than adopts a set of clients when the clients ask for being adopted. A peer obtains a server list from specific Web sites and it chooses one of the
servers for accessing to the network. Each server has defined a limit of adopted clients and will reject clients when the list is full.

### 3.3 Database and information representation

There are several ways of storing and representing discovery information across the network:

- **General databases**, which stores information in database entries. This can be a fast solution but not the optimal.

- **Directories**, such as LDAP directories. Here design premises are focused on providing high reliability, scalability and performance for many simple operations, in opposite to distributed relational databases schemes which are focused on satisfying complex operations \[27\].

- **Hash based techniques**:
  - Hash tables, or hash map, involve a data structure that associate keys with values. The primary supported operation is efficient lookups: given a key (e.g., a *Topic* in DDS), find the corresponding value (the *Endpoints* associated list). The best performance of hash tables is obtained when dealing with arbitrary lookups and the worst performance is obtained for sequential key retrieving. While the first is found normaly in DDS, the second is uncommon operation in DDS.
  - Distributed Hash Tables (DHT), explained below.
  - Bloom filters are explained deeply in Chapter \[4\]. They are used in many database applications to reduce the information stored or alternatively to represent database content. Bloom filters are used in Pastry for optimization purposes and in OSDA for many cases, for example, when checking local cache in local discovery protocol.

### 3.4 Distributed Hash Tables, DHT

The problem of scalability in *Gnutella* project motivates the design of several alternatives in pure decentralized P2P networks, most of them are based on *Distributed Hash tables* (DHTs).

Hari Balakrishnan et al. \[29\] define *Distributed Hash tables*:

The recent algorithms developed by several research groups for the lookup problem present a simple and general interface, a *distributed hash table* (DHT).
3. Discovery technologies review

Data items are inserted in a DHT and found by specifying a unique key for that data. To implement a DHT, the underlying algorithm must be able to determine which node is responsible for storing the data associated with any given key. To solve this problem, each node maintains information (e.g., the IP address) of a small number of other nodes (“neighbors” in the system, forming an overlay network) and routing messages in the overlay to store and retrieve keys. […] The distributed hash table, for example, is increasingly finding uses in the design of robust, large-scale distributed applications. It appears to provide a general-purpose interface for location-independent naming upon which a variety of applications can be built. Furthermore, distributed applications that make use of such an infrastructure inherit robustness, ease of operation, and scaling properties. A significant amount of research effort is now being devoted to investigating these ideas.

In order to improve fault tolerance, these systems use replication to solve nodes failures and to ensure data availability. Caching is used to make temporary copies and to balance load in data requests. The problem of finding resources is solved in a deterministic way, in other words, if a resource is in the net, then it can be obtained [30]. This is not ensured in ed2k networks because of the success in discovery process is conditioned by the server list that the client has.

DHT are one of the services provided in overlay P2P, also known as Key Based Routing (KBR) networks, because the message routing is based on node keys. A short definition of KBR is [31]:

Key based routing (KBR) is a lookup method used in conjunction with distributed hash tables (DHTs). While DHTs provide a method to find a host responsible for a certain piece of data, KBR provides a method to find the closest host for that data, according to some defined metric. This may not necessarily be defined as physical distance, but rather the number of network hops. KBR improves the efficiency of decentralized information retrieval in peer-to-peer networks.

Different projects have implemented DHT over P2P networks such as Chord, Symphony, Pastry or Kademlia, often designed to deal with large dynamic peer-to-peer networks with frequent node arrivals and departures. In these implementations the nodes are grouped in a structured way, normally constituting a ring topology to ensure resources location. To join the network it is only necessary to know a node. Pastry, Chord and other related schemes provide a public API to access the DHT.

Nowadays, the Kad network, based on Kademlia, is working parallel with ed2k network with hundreds of thousands of users. However, we do not know if this success is due to the information shared about nodes with the parallel ed2k network.
3.5 Replication methods

There are several replication methods. Some topologies implement replication implicitly, typically DHT solutions. Others need to be configured in an explicit way to do replication, such as LDAP directories.

From now on, the term “server” refers to as a “SuperNode” or a “Directory Facilitator” or other discovery entities as well.

3.5.1 Complete information replication in servers

In this case the information is copied in every SuperNode and the updates are propagated between SuperNodes. For example, in DFSDA, the first Service Directory Agent (SDA) which receives new information sends it using multicast to every other SDA and associated Directory Facilitator (DF). Depending on the application, concurrency access can be a problem for consistent replication. However, classic problems in distributed databases are not so severe in DDS discovery protocol because each Participant is the only one that produces information about itself. Then, the sending Participant is the only one who sends its information to the SuperNodes. Therefore, it is not possible to write the same information from different sources.

The complete information replication has some well known limitations. The replication entities have a limit for storage, processing and network resources. These problems can be overcame with complementary solutions, for example:

- Using Bloom filters to reduce the storage and network traffic.
- The workload of each server can be divided up using, Round Robin.

3.5.2 Partial information replication in servers

Other way of replication is achieved by splitting information in segments in order to replicate it. In this case, the network topology is strongly related to the way that the partial replication is implemented.

For example, in Chord’s ring each node can store information of other nodes by replicating some neighbours of a node. If a node gets down, its neighbours have the enough information to maintain the information consistence in the ring and, therefore, a recovery algorithm can be applied. Recovery or stabilization algorithms are usually complex. This replication is implicitly implemented in DHT’s based solutions.

In tree or hierarchical topologies the replication is done adopting branches or hierarchical structures. Each considered part is completely replicated in one or several nodes. In systems such as LDAP directories this mechanism is explicitly defined by an administrator when configuring the servers.
3. Discovery technologies review

3.6 Learning, cache and distinguished local and global discovery

We consider three related terms which are learning, cache and locality. We talk about learning and cache as ways for interchanging discovery information between different entities. The locality will refer to the consideration of building the topology considering some criterion to cluster nodes in order to improve some aspects, like minimizing hops in message passing or using local discovery.

Nodes can share information about discovery and routing to optimize searches and reduce routing latency. The information shared between nodes can be related to nodes discovery and nodes content. DFSDA and OSDA implement both of them.

The cache and learning in nodes seems to be related to the functional relationship between the elements of the network:

1. In Client-Server relationship the clients only known discovery servers and do not interchange information with other clients. To reduce server’s load and network traffic it is common to install in each client some cache services, such as the Name Service Cache Daemon (NSCD) in Unix-like operating systems.

2. In centralized peer-to-peer there two main alternatives:

   (a) Each node uses only servers to do discovery.

   (b) Each node can interchange discovery information also with other nodes which are not discovery entities. For example, in ed2k network you can enable your client to interchange this information from “friend nodes”, and add the results to the server’s responses.

3. In pure P2P systems learning between nodes is indispensable and it is implicitly defined in every algorithm implementation.

In what follows we review some ways in which learning is implemented.

In OSDA and DFSDA local and global discovery are clearly separated into different protocols and entities. Before doing global lookups, local lookups are performed. If the local search does not produce results, or the results are obsoletes, a global lookup is performed. These searches are directed to a group of neighbours nodes and/or to a special entity. In OSDA, these groups are called Domains. The special entity is used to manage all the external discovery communications in a group. Some names for these entities are “Directory Facilitator” in DFSDA, “broker” in OSDA, etc.

The way in which OSDA and DFSDA do the global discovery can be easily ported to RTDDS and the Publication/Subscription model. If local discovery is allowed, to use cache in nodes or similar techniques is usual. This is the case for DHT’s implementations.
and OSDA and, optionally, in DFSDA. Normally, in DHTs each node stores a set of discovered entities and resources. Also in Chord, if a node does not know where a resource is, it can answer the location of someone else who has the resource or how knows where the resource is.

In DFSDA’s groups there are stored local and remote discovered services, allowing nodes in a group to ask each other. OSDA summarizes cache in a Domain with a counting Bloom filter, which is managed by the broker. The broker sends the filter periodically to the nodes in a Domain. Therefore, these nodes can check if a resource is in local cache using the Bloom filter.

### 3.7 Distribution of entities and resources announcements

In DFSDA’s [19] two main mechanisms are identified for entities or resources announcements to be used in P2P and in agent environments:

Accepting that alternatives to the centralized approach are required, we consider two alternative approaches to distributing service announcements:

- The “Push” solution, in which a device that offers a service sends unsolicited advertisements, and the other devices listen to these advertisements selecting those services they are interested on.
- The “Pull” solution, in which a device requests a service when it needs it, and devices that offer that service answers the request, perhaps with third devices taking note of the reply for future use.

The DFSDA [19] solution adopts both techniques and takes advantage of them. The “Push” and “Pull” division is studied in other works with different terminology.

### 3.8 Comparison table in discovery

Tables [3.1] and [3.2] summarizes features in studied platforms.

### 3.9 Performance and scalability

Since pure P2P systems were proposed, several papers have studied real P2P performance and scalability. We briefly comment the main conclusions.

*Understanding Chord Performance and Topology-aware Overlay Construction for Chord* [32], by Li Zhuang and Feng Zhou, studies performance of the Chord scalable lookup system
3. Discovery technologies review

<table>
<thead>
<tr>
<th>Alternative</th>
<th>LDAP directories</th>
<th>ed2k</th>
<th>Gnutella</th>
<th>Kademlia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>Client-Server</td>
<td>Centralized P2P</td>
<td>Pure P2P</td>
<td>Pure P2P</td>
</tr>
<tr>
<td>relationship</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network topology</td>
<td>Tree</td>
<td>Unstructured</td>
<td>-</td>
<td>Ring</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client/Node</td>
<td>A client knows from one to many servers but it do not determine the lookup capacity</td>
<td>A client knows a server or a server list and it conditions the searches results</td>
<td>A client knows a group of nodes</td>
<td>A client knows a group of nodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node learning and cache</td>
<td>Optional with NSCD</td>
<td>The clients can interchange resources information</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>Can communicate with other servers. It attends clients or servers petitions</td>
<td>Adopts a set of clients</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server learning and cache</td>
<td>Can forward petitions</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initialization</td>
<td>The client knows one or more servers</td>
<td>The client has a list of servers available.</td>
<td>The client need to know where is a node of the network</td>
<td>A client knows a node</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other entities</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.1: Features in discovery platforms, part 1

with several performance enhancements. According to the authors, the main criticism is that the location cache implementation is very effective in static networks, but it does not scale to more than 2000 nodes in a typical file-swapping network setting. This is because of cache entry validity is not active, a node tends to have more and more stale cache entries over time. This results in time-outs and redundant hops.

Performance of Pastry in a Heterogeneous System [33] studies how Pastry performs in a heterogeneous network environment of varying size from 30 up to 3000 nodes. The large traffic overhead for management traffic makes the overlay nonfunctional if it grows too much. The main conclusion is that it is needed to avoid the creation of bottlenecks in the routing system. In Pastry, this is the management traffic overhead in the overlay network. The authors try to do it by partitioning the routing tables at the cost of increased path lengths and response times. Unfortunately, with 3000 nodes in the network and a workload of 100,000 messages they observed significant performance degradations when partitioning the routing tables.

3.10 Related work review summary

Up to the author’s knowledge, there is no current research papers related to DDS discovery. Moreover, the discovery issue is a research topic for many different environments which include network operating systems [27], mobile communications [34], agents platforms [19], the peer-to-peer networks and so on. Therefore, it is important to review dis-
3.10. Related work review summary

<table>
<thead>
<tr>
<th>Alternative</th>
<th>Chord/Pastry</th>
<th>OSDA</th>
<th>PDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional relationship</td>
<td>Pure P2P</td>
<td>Pure P2P for global directory</td>
<td>Hierarchical P2P</td>
</tr>
<tr>
<td>Network topology</td>
<td>Ring</td>
<td>Chord’s ring for global index and a hierarchical topology for the rest of entities</td>
<td>Hierarchical topology. Discovery entities have to do multicast to communicate each other</td>
</tr>
<tr>
<td>Client/Node</td>
<td>The client knows a set of nodes before and after its in the ring</td>
<td>The clients are grouped and coordinated and globally represented by a broker</td>
<td>The Service Agents (SA) only manages information of interest</td>
</tr>
<tr>
<td>Node learning and cache</td>
<td>Yes</td>
<td>Yes. The local cache is summarized in a BF for performance improvement</td>
<td>Each group has a list of discovered remote services</td>
</tr>
<tr>
<td>Server</td>
<td>-</td>
<td>Global P2P index in a Chord’s DHT</td>
<td>The Directory Facilitator (DF) supplies discovery information to SA</td>
</tr>
<tr>
<td>Server learning and cache</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Initialization</td>
<td>The client need to know where is a node of the network</td>
<td>A Client join to a group managed by a broker</td>
<td>A Client must be adopted by a DF</td>
</tr>
<tr>
<td>Other entities</td>
<td>Service Broker, Policy Server</td>
<td>The Service Discovery Agents (SDA) coordinate DF discovery information</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Features in discovery platforms, part 2

discovery related techniques before trying to improve the current DDS discovery process.

Initially, discovery techniques should consider functional relationship between the network nodes. The first approach is the client-server architecture which has well known problems. Moreover, during the last years more paradigms has been introduced: the centralized peer-to-peer, pure peer-to-peer and hierarchical peer-to-peer[17]. DDS can be placed into the pure peer-to-peer class.

Peer-to-peer architectures can be divided into structured and unstructured systems. We classify DDS as the unstructured P2P category. Rather than organize nodes and DDS entities into a structure, such as Distributed Hash Tables (DHT) implemented in Chord [15], Pastry [16] and others, unstructured communication paths are built by the DDS Topic sets.

As mentioned in Section [3.7] in P2P architectures there are two approaches of distributing discovery information: the “Push” solution and the “Pull” solution. In the “Push” solution a node, or entity in a node, sends unsolicited advertisements to other nodes. The “Pull” option explicitly sends information request to nodes in the network. For example, the first DHTs implementations, such as Chord and Pastry, used a “Pull” strategy for information distribution. In those systems, nodes demand keys information
to neighbour nodes which answer or forward the requests. The neighbour nodes can answer the information received requests or forward it to other nodes. Recent works show that the “Push” alternative works better in some environments [21]. Specially in Publication/Subscription architectures the announcements approach is used.

Reviewing the discovery related technologies we found the Bloom filters [11]. Bloom filters are used both in research and industry fields for summarizing content. An excellent review of network applications of Bloom filters can be found in A. Broder and M. Mitzenmacher [35]. Bloom filters will be studied in Chapter 4.
This chapter presents an introductory overview of Bloom filters (BF) and its variations, as well as a review of some of its main applications. The Chapter also includes a discussion about the interest of using some Bloom filter variations.

4.1 Introduction and original Bloom filters

The Bloom filter, conceived by Burton H. Bloom in 1970 [11], is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. It supports element insertion operation but not element deletion (though this can be addressed with counting filters approach as further explained in Section 4.2).

A filter is a $m$ length bit array that initially contains only zeros. A set of $k$ hash functions maps elements to one position in the array. The insertion operation consist on to set to one the array positions given by the $k$ hash functions. Similarly, the key membership test operation consists on to check the result of hash the key and checking whether the proper positions are set to ones in the array. If anyone of the positions is set to zero the item does not belong to the set. Figure 4.1 shows an example of how a Bloom filter works.

The cost of this extremely compact data representation is that membership queries can produce false positives, but no false negatives. There is a probability that a set of array positions is set to ones and the element is not in the set. This probability depends on the number of hash functions used ($k$), the size of the array ($m$) and the number of items represented in the filter ($n$). The probability of obtaining a false positive in a membership query can be expressed as [36]:

$$f = (1 - (1 - \frac{1}{m})^k)^k \approx (1 - e^{-\frac{kn}{m}})^k$$  \hspace{1cm} (4.1)
4. Bloom filters

Given a set $S = \{x_1, x_2, x_3, \ldots, x_n\}$ the problem is to answer queries of the form: Is element $y$ in the set $S$?

Given a

- Vector $v$ of size $m$
- $k$ hash functions, $H_i(x)$
- A set $S$ with $n$ elements

Start with an $m$ bit array, filled with 0s.

Bit Vector $v$

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Element $a$

$H_1(a) = P_1$

$H_2(a) = P_2$

$H_3(a) = P_3$

$H_4(a) = P_4$

Hash each item $x_j$ in $S$ $k$ times. If $H_i(x_j) = p$, set $V[p] = 1$.

To check if $y$ is in $S$, check $V$ at $H(y)$. All $k$ values must be 1.

Possible to have a false positive; all $k$ values are 1, but $y$ is not in $S$.

Source: Michael Mitzenmacher, “Codes, Bloom Filters, and Overlay Networks”

Figure 4.1: Bloom filters operations description
4.2 Counting Bloom filters

The basic ideas for applying Bloom filters are: in a system in which a given entity needs to know exactly what items other entities have, the traditional message should be *give me the list of items you have*. But if the list becomes enough long, this approach becomes not very flexible. Using Bloom filters we can obtain approximate lists by moving to paradigms like *give me information to deduce what items do you have*.

In general terms, BF provides an answer in:

- “Constant” time (time to hash).
- Small amount of space.
- But with non-zero probability of being wrong.

The Bloom filters have the following interesting properties:

- Unlike sets based on hash tables, any Bloom filter can represent the entire universe of elements. In this case, all bits are 1. Another consequence of this property is that add never fails due to the data structure “filling up”, although the false positive rate increases steadily as elements are added.

- Union and intersection of Bloom filters with the same size and set of hash functions can be implemented with bitwise OR and AND operations, respectively.

In what follows, a short review of some BF variants is summarized.

### 4.2 Counting Bloom filters

*Counting Bloom filters* provide a way to implement delete item operations without recreating the filter. In counting filters the array positions (buckets) are extended from being single bits to n-bit counters. In fact, regular Bloom filters can be considered as counting filters with a bucket size of one bit. Counting filters were introduced in Summary cache [36].

The insert operation is extended to increment the value of the buckets and the lookup operation checks that each of the required buckets is non-zero. The delete operation, obviously, then consists of decrementing the value of each of the respective buckets.

Arithmetic overflow of the buckets is a potential problem, therefore buckets should be large enough to make this even rare. If it does occur, then the increment and decrement operations must leave the bucket set to the maximum possible value in order to retain the properties of a Bloom filter.

As well as in Summary cache (used in Squid [37] Cache Digest), counting filters are used in OSDA [17] for creating a two-level indexing scheme that combines counting...
4. Bloom filters

Bloom filters with the Chord’s Distributed Hash Table. In both projects, the problem of estimating the appropriate dimension of the filter is mentioned.

Squid authors mention that Bloom proved that the optimal filter utilization is 50% (half of the bits are off), but they do not used any formula to estimate a priori the number of entries.

In Distributed Search in Semantic Web Service Discovery [18], by Joanna Irena Ziembicki, we find very interesting annotations for using counting (or “cumulative”) Bloom filters in practice. In Section 7.4.5 the problem of false positives are explicitly addressed:

A cumulative Bloom filter can handle only a limited number of insertions before it becomes imprecise and admits a high number of false positive results to queries. Since the probability of false positives can be calculated based on the size of the filter and the number of item insertions (See Section 2.4.1), the simple solution is to limit the number of advertisements allowed in one filter according to a set threshold of probability of false positives (for example, 1%). Then, a new cumulative filter can simply be added when the old one is “full” – the advertisement process can easily accommodate two or more cumulative category Bloom filters from the same service provider. [18]

Also, in section 8.4 they say:

Since we use a soft-state approach, where service advertisements expire over time, we need to use counting Bloom filters to avoid stale entries. Counting Bloom filters, however, use more space than regular Bloom filters. To preserve space, we use non-counting Bloom filters for summarizing single service descriptions within the local index, and counting Bloom filters for cumulative summaries sent to the global index. [18]

The mentioned problems have been overcome with general solutions by recent variations such as Dynamic Bloom filters or Scalable Bloom filters.

4.3 Generalized Bloom Filter

The Generalized Bloom Filter (GBF) are a space-efficient data structure to securely represent a set. Different from the standard Bloom Filter, the GBF imposes an upper bound on the false positive probability. The key idea of the GBF is to reset bits of the filter. For that purpose, the GBF uses both hash functions that set and hash functions that reset bits. This procedure limits the false positives at the expense of introducing false negatives in membership queries [38].

An implementation of GBF is available in Rafael P. Laufer’s Homepage [39].
Therefore, the GBF ensures bounded false positives and false negatives rates. Unfortunately, the existence of false negatives is a disadvantage in discovery strategies. False negatives implies that using only GBF we can not find entities in a network in a deterministic way, understanding deterministic as if something is in the network we always find it. To deal with false-negatives we should need to use additional solution to check if a response is a false-negative or if the desired Endpoint is really not present.

4.4 Compressing Bloom Filter

Compressed Bloom filters have been introduced to improve performance when the Bloom filter is passed as a message and its transmission size is a limiting factor. This is translated into fast message transmissions and, in consequence, the possibility of increasing the filter size to reduce false positive probability. The cost is the processing time for compression and decompression and the additional requirement of having more memory in nodes that stores the compressed Bloom filters [40].

In [40] the author asserts that advantages of Bloom filters affect to original Bloom filters and counting Bloom filters, and he expects that similar variations of Bloom filters would benefit from compression as well.

It seems that filters are not going to be enough bigger in SDP-Bloom to benefit from compressed Bloom filters. The alternatives mentioned before, such as sending just Endpoints information instead of the entire Bloom filter can reduce the network traffic. However, discovery alternatives which include Servers or SuperNodes for Participant adoption can benefit from Compressed bloom filters.

4.5 Bloomier filters

While Bloom filters allow membership queries on a set, the Bloomier filter [41] generalizes the scheme to a data structure that can encode arbitrary functions. That is, Bloomier filters allow us to associate values with a subset of the domain elements in an associative array. A false positive is defined as returning a result when the key is not in the map and the map will never return the wrong value for a key that is in the map. In addition, they allow for dynamic updates to the function, provided that the support of the function remains unchanged.

The method performs fairly well in those situations where the function is designed only over a small portion of the domain, which is a common occurrence.

The uses of Bloomier filters can give us more possibilities and do not depend of any mechanism to deal with false positives. However, public implementations are not available.
4. Bloom filters

4.6 Spectral Bloom filters

The Spectral Bloom Filters (SBF) are an extension of the original Bloom Filter to deal with multisets. SBF allows the filtering of elements whose multiplicities are below a given threshold at query time. Using memory sizes slightly larger than the original Bloom Filter, SBF supports queries on the multiplicities of individual keys with a guaranteed, small error probability. The SBF also supports insertions and deletions over the data set.

In [42] some methods are proposed to reduce the probability and the magnitude of errors. Also, it is included an efficient data structure as well as algorithms to build it incrementally and maintain the filter over streaming data.

Some applications of SBF are:

1. Aggregate queries over specified items, that is, queries like:
   ```sql
   SELECT count(a1) FROM R WHERE a1 = v
   ```

2. Adhoc iceberg queries. Iceberg queries compute one or more aggregate functions to find aggregate values above a specified threshold. A typical iceberg query would be:
   ```sql
   SELECT a, count(a)
   FROM tab
   GROUP BY a
   HAVING count(a) = 100;
   ```

   This base form can easily be made more complicated by doing self joins and the like. We call such queries iceberg queries because of the number of above-threshold results is often very small (the tip of an iceberg), relative to the large amount of input data (the iceberg).

3. Spectral Bloom joins. Bloom joins are a method for performing a fast distributed join between relations R1 and R2 residing on different database servers - R1 in site 1 and R2 in site 2, based on attribute a.

4.7 Dynamic Bloom Filter

Counting Bloom Filters (CBF) have been proposed to overcome the problem of deletions and insertions on multisets over time, not supported by original BF. Dynamic Count Filters (DCF) are presented as a new dynamic and space-time efficient representation of CBF. Although DCF does not make a compact use of memory, they are faster and more efficient than any previous proposal, independently of the incoming data characteristics [43], in terms of memory consumption.
The DCF structure captures the best of SBF and CBF. They provide fast CBF access times and the SBF adaptivity to changing data patterns. Unlike SBF, DCF does not require the use of indexes and, in consequence, they reduces the amount of memory requirements in many cases.

In [43] CBF, SBF, and DCF qualitative comparisons are established:

<table>
<thead>
<tr>
<th>Filter</th>
<th>Counters size</th>
<th>Access Time</th>
<th>#Rebuilds</th>
<th>Saturated counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBF</td>
<td>Static</td>
<td>fast</td>
<td>not allowed</td>
<td>Yes</td>
</tr>
<tr>
<td>SBF</td>
<td>Dynamic</td>
<td>slow</td>
<td>high</td>
<td>Eventually</td>
</tr>
<tr>
<td>DCF</td>
<td>Dynamic</td>
<td>fast</td>
<td>low</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 4.1: Qualitative Comparison of CBF, SBF, and DCF

In general terms, it seems that SBF and DCF allow new applications of Bloom filters. We must take both in mind for future applications.

### 4.8 Scalable Bloom Filter

One of the limitations of original Bloom filters is that the size of the filter must be defined a priori based on the number of elements to store and the desired false positive rate. For this reason, it is impossible to store extra elements without increasing the false positive probability. Scalable Bloom Filters [44], are a variant of Bloom filters that can adapt dynamically to the number of elements stored, while assuring a maximum false positive probability. The mechanism adapts to set growth by using a series of classic Bloom filters of increasing sizes and tighter error probabilities, added as needed. The two key ideas are:

- A Scalable BF is made up of a series of one or more (plain) Bloom filters; when filters get full, due to the limit on the fill ratio, a new one is added; querying is made by testing for the presence in each filter.

- Each successive bloom filter is created with a tighter maximum error probability on a geometric progression, so that the compounded probability over the whole series converges to some wanted value, even accounting for an infinite series.

Memory space of Scalable BF are logically bigger than BF, moreover, if the filters used are not so bigger, the process of rebuilding the filter with a different size can be an easier and a faster solution than using Scalable BF.

Due to the novelty of Dynamically and Scalable Bloom filters, there are not many reviews about its, but the idea of making BF flexible in all senses is always interesting.
4. Bloom filters

4.9 Bloom filters related resources

In this section resume some interesting resources related to Bloom filters apart from scientific references.

Network applications of Bloom filters

The paper *Network Applications of Bloom Filters: A Survey* by A. Broder and M. Mitzenmacher is an excellent overview. An updated version is available online at M. Mitzenmacher’s Homepage [45].

Bloom filters dimension estimation

We find a simple way of estimating the size of Bloom filters in the article *Using Bloom Filters* by Maciej Ceglowski[46].

The false positive rate can be calculated for any filter by using the formula:

\[ f = (1 - e^{-kn/m})^k \tag{4.2} \]

Where \( c \) is the false positive rate, \( k \) is the number of hash functions, \( n \) is the number of keys in the filter, and \( m \) is the length of the filter in bits.

When using Bloom filters, we very frequently have a desired false positive target rate and we are also likely to have a rough idea of how many keys we want to add to the filter. We need some way of finding out how large a bit vector assures that the false positive rate never exceeds our limit. The following equation will give us vector length from the error rate and number of keys:

\[ m = \frac{-kn}{\ln(1 - c^{1/k})} \tag{4.3} \]

To show how error rate and number of keys affect the storage size of Bloom filters, Table 4.2 lists some sample vector sizes for a variety of capacity/error rate combinations.

Further lookup tables can be found for various combinations of error rate, filter sizes, and number of hash functions at *Boom Filters - the math* in the original paper of Summary Cache [36].

### 4.9. Bloom filters related resources

<table>
<thead>
<tr>
<th>Error Rate</th>
<th>Keys</th>
<th>Required Size</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>1K</td>
<td>1.87 K</td>
<td>1.9</td>
</tr>
<tr>
<td>0.1%</td>
<td>1K</td>
<td>2.80 K</td>
<td>2.9</td>
</tr>
<tr>
<td>0.01%</td>
<td>1K</td>
<td>3.74 K</td>
<td>3.7</td>
</tr>
<tr>
<td>0.01%</td>
<td>10K</td>
<td>37.4 K</td>
<td>3.7</td>
</tr>
<tr>
<td>0.01%</td>
<td>100K</td>
<td>374 K</td>
<td>3.7</td>
</tr>
<tr>
<td>0.01%</td>
<td>1M</td>
<td>3.74 M</td>
<td>3.7</td>
</tr>
<tr>
<td>0.001%</td>
<td>1M</td>
<td>4.68 M</td>
<td>4.7</td>
</tr>
<tr>
<td>0.0001%</td>
<td>1M</td>
<td>5.61 M</td>
<td>5.7</td>
</tr>
</tbody>
</table>

Table 4.2: Some sample vector sizes for a variety of capacity error rate combinations in Bloom filters

**Bloom filter calculator**

An online Bloom filter calculator is available in Panagiotis (Pete) Manolios’s homepage[^2]. In this calculator we can specify any combination of 2 or 3 arguments:

- $m$, the number of bits in the Bloom filter.
- $n$, the number of elements inserted into the Bloom filter.
- $k$, represents the number of hash functions used.
- $p$, denotes the false positive rate.

**Bloom free available implementations**

We have access to the following Bloom filters implementations available under the GPL license:

- Bloom Filters C++ implementation, available in General Purpose Hash Function Algorithms by Arash Partow[^3].
- *A Fast, Flexible and Extensible Implementation of Bloom Filter[^4]* in Java language, by Hongbin Liu’s.

[^3]: http://www.partow.net/programming/hashfunctions/index.html
[^4]: http://grids.ucs.indiana.edu/~lhb/software/bloomfilter.html
4. Bloom filters

4.10 Bloom filter variants discussion

Since the initial publication of Bloom filters, a number of variants based on the original idea have been proposed to extend the initial features. In this section we examine if a Bloom filter variant should be used for DDS discovery for adding features for the original Bloom filter.

4.10.1 Items deletion support

Support of item deletion from a given set without rebuilding the entire Bloom filter can be advantageous. Firstly, the CPU load will be reduced and, hopefully the network traffic will also be reduced. For instance, once the network has passed the initialization phase, if an item represented in the filter is deleted a deletion command can be sent to the network for updating remote filters. Often a element data size is significantly smaller than the filter size. Counting Bloom filters \[36\] supports deletion at the expense of increasing the data representation size in the filter.

4.10.2 Filter resizing and information updates

The proposed algorithm does not take into account how to update changes or how to send new discovery information to all Participants. Unfortunately, the size of the filter increases with the number of Entities in each Participant. Depending on the particular conditions, it could be better to send the information updates instead of the entire filter. The update problem is studied in the Summary Cache \[36\] scheme, but none of the alternatives is clearly chosen.

The false positive rate increases with the number of elements inserted in the filter. Is desirable to maintain the false positive probability under a reference value. Typically the filter bits array can be resized for decreasing this probability when the number of elements grows. Dynamic \[43\] and Scalable Bloom \[44\] filters allow to grow the filters withoutincrementingthefalsepositiverateandavoidingthefilterrebuildoperation. This, the filters can be resized if the number of items grows to maintain the false positive ratio.

4.10.3 Conclusion

We assume that the major discovery traffic load occurs during the initialization of the DDS platform. Implicitly, it is assumed that most DDS entities are intended to be permanent along the time. This is, once a DDS system is initialized is not frequent to delete Entities.

Regarding to items deletions feature, the implementation of Counting Bloom filters is simple. Moreover, the complexity that must be added to the protocol for supporting item
deletion and filter size incrementation does not justify the use of Counting Bloom filters.

Finally, our experience with original Bloom filters and the number of items we manage per filter is that we can keep a small false positive ratio without incrementing so much the filter size. Therefore, the complexity cost of attenuating false positives does not justify the use of variations such as the Bloomier filters. The same reasons can be applied to the filter resizing features: the filter is small enough than the filter build cost does not justify the use of scalable Bloom filters.
4. Bloom filters
5.1 Overview

Section 2.2.2 characterizes analytically the SDP protocol. According to the presented equations, for those scenarios with a big number of Participants and Endpoints in each Participant, we identify two problems that must be overcome:

1. Memory requirements. Each Participant does not know what other Participants have before interchanging and storing the complete remote list of Endpoints. Therefore the memory requisite grows with the number of Participants and Endpoints because each Participant stores information of every entity, even for entities not interested in.

2. Network traffic. To obtain the whole Endpoints lists from all Participant generates a lot of traffic, specially if multicast is not available. The traffic grows regarding to:

   - Number of messages sent to the network.
   - Bandwidth consumed by the discovery information representation on the wire.

To deal with these problems we propose to represent most of the discovery information by modifying SDP for using Bloom filters. Therefore each Participant will send its own Bloom filter which epitomises its Endpoint set to other Participants in the same DDS Domain. Consequently, the number of messages sent for announcing the Endpoints is reduced to a unique message containing a Bloom filter. The memory requirements and network traffic load will be decreased because of the Endpoint information is condensed by the Bloom filter. In this way, we are changing the SDP dialogue paradigm between Participants from “give me all information you have” to “give me information to know what you have”. We call this alternative SDP Bloom.
5. SPD with Bloom filters

Bloom filters allow each Participant to check if any of its Endpoints of interest is in the set represented by the filter. In SPDBloom each Participant continues storing information of all entities but with significantly smaller size.

The keys or items stored in the filter must be any unique identification for Endpoints. We can use just the Topic name as a key to be inserted into the filter. In this case a Participant will perform membership queries to the filter by using the Topic name of the local Endpoints. Also, we can build a more complex key including the type name and the type code. Therefore the Participant will check a key composed by the union of three elements (the topic name, the type name and the type code) against the filter. The alternatives and influence of the key composition are discussed in Chapter 6.

5.2 Algorithm Description

The first design decision is to decide when the Bloom filter should be sent to other Participants. The OMG RTPS [10] standard divides discovery protocols in the PDP and EDP. Thinking in the purpose of each protocol, Endpoints information should be included in the EDP protocol, this is, the filter should be included as discovery information in the EDP. However, we consider to include the filter in the Participant DATA messages which are sent periodically to keep the liveliness of the DomainParticipant in the Participant discovery procedure. In this case, the SEDP will be adopted as the Endpoint discovery procedure for SDPBloom but it will be only started if there is a match in the Bloom filter.

Sending the filter during the PDP has two advantages. First, this alternative is closer to the content announcement policy. Second, this alternative is more efficient in terms of number of messages sent to the network. This is, for announcing its presence and its Endpoints filter, a Participant will send $P$ number of messages to the network, where $P$ is the number of Participants in the network. This means one message for each Participant in a Domain. In the other hand, to send the filter after the PDP would imply an extra message sent to the network for announcing the filter, so the total messages for announcing the Participant and its Endpoints would be $2 \times P$. Therefore, from now on it will be assumed the first alternative.

If the Bloom filter is sent with the Participant advertisement message, the SPDBloom will work like this:

1. When enabling the new Participant, a Bloom filter is created for representing its Endpoints.
2. Each time a new local Endpoint is created and enabled, it is added to the filter.
3. The Participant starts sending periodically its Participant DATA which includes the filter. In the same way, all remote Participants will send its Participant DATA accordingly.
5.2. Algorithm Description

Figure 5.1: SDP-Bloom algorithm description
5. SPD with Bloom filters

We omit PDP information. The BF is sent in a refresh period defined as a QoS parameter. The light blue shows the suppressed traffic.

![SDP-Bloom diagram](image)

Figure 5.2: SDPBloom nodes dialog

4. The Participant stores all new discovered Participant filters and tries to know if the desired remote Endpoints belong to each received filter.

5. For each desired Endpoint which matches with a remote Participant’s filter, the Participant tries to start a publication or subscription. This means the SEDP data will be interchanged between two Participants. Instead of sending all Endpoints information only matched Endpoints data will be sent.

The proposed algorithm is illustrated in Figure 5.1. Figure 2.2 describes a SDPBloom nodes dialogue example. This second diagram shows the both SDP and SDPBloom network messages, hence the saved number of messages sent to the network can be appreciated. The filter rebuild period (if there are changes in the entities) and sent period are also shown. This event can occur after a specified period or can be on-demand after a Participant DATA message reception.
5.3 SPDBloom Scalability Analysis

In this section preliminary evaluation is estimated for the proposed SPDBloom algorithm. For easier comparison, we present the analysis extending the SDP scalability analysis in Section 2.2.2. We include a new variable called Matched Endpoints \( ME \) for representing the average local Endpoints matched with remote Endpoints for each Participant. After a matching of one or several Endpoints, a set of information must be transferred for starting the publication/subscription. This information is the same that current SEDP defines for configuring the communication as explained in Section 2.1.

We will include \( ME \) in equations for representing that this information is transferred. Moreover, we will not include \( ME \) for analytical data generation because of two reasons. The first one, is because that information can be considered as information used for configuring and establishing a connection between peers, therefore, it is not strictly discovery information. The second reason is that in large scenarios \( ME \) can be considered as not significant metric since \( ME \ll E/P \). However, we keep the \( ME \) metric in some equations for future evaluations of these data relevance.

In a SDPBloom, each Participant only sends and receives a filter representing each other Participant in network, therefore:

\[
N_{\text{Participant}} = 2 \times (P - 1) \times ME \sim 2 \times P \times ME 
\]  
(5.1)

The total sent messages is equal to the number of Participant multiplied by what it is sent by each Participant, that is, one Bloom filter:

\[
N_{\text{total}} = P \times (P - 1) \sim P \times P \times ME 
\]  
(5.2)

There is only one message per Participant using multicast, therefore:

\[
N_{\text{MParticipant}} = 1 + (P - 1) = P \times ME 
\]  
(5.3)

The total number of sent messages is:

\[
N_{\text{Mtotal}} = P \times P = P \times ME 
\]  
(5.4)

The needed storage for each Participant is one Bloom filter plus its Endpoints and its Matched Endpoints:

\[
M_{\text{Participant}} = P + E/P + ME 
\]  
(5.5)

The storage needed to keep the live transport sessions is equal to SDP:

\[
S_{\text{Participant}} = (P - 1) \sim P 
\]  
(5.6)

If a new empty Participant is added to the network, the number of generated messages is independent of \( E \):

\[
N_{\text{marginalParticipant}} = 2 \times P 
\]  
(5.7)
5. SPD with Bloom filters

<table>
<thead>
<tr>
<th>Scenario name</th>
<th>Participants (P)</th>
<th>Topics (T)</th>
<th>Endpoints (E)</th>
<th>E / P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small System</td>
<td>100</td>
<td>400</td>
<td>2000</td>
<td>20</td>
</tr>
<tr>
<td>Medium System</td>
<td>1000</td>
<td>1000</td>
<td>20000</td>
<td>20</td>
</tr>
<tr>
<td>Large System</td>
<td>10000</td>
<td>9100</td>
<td>200000</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 5.1: Small, medium and larger scenarios description

<table>
<thead>
<tr>
<th>Discovery Protocol</th>
<th>Np</th>
<th>Nt</th>
<th>Mp</th>
<th>Sp</th>
<th>Ap</th>
<th>Ae</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDP unicast small</td>
<td>4000</td>
<td>200,000</td>
<td>2100</td>
<td>100</td>
<td>200</td>
<td>103</td>
</tr>
<tr>
<td>medium</td>
<td>40,000</td>
<td>20,000,000</td>
<td>21,000</td>
<td>1000</td>
<td>22000</td>
<td>1005</td>
</tr>
<tr>
<td>large</td>
<td>400,000</td>
<td>2,000,000,000</td>
<td>210,000</td>
<td>10000</td>
<td>220000</td>
<td>10011</td>
</tr>
<tr>
<td>SPDBloom small</td>
<td>200</td>
<td>10,000</td>
<td>120</td>
<td>100</td>
<td>200</td>
<td>103</td>
</tr>
<tr>
<td>medium</td>
<td>2000</td>
<td>1,000,000</td>
<td>1020</td>
<td>1000</td>
<td>2000</td>
<td>1005</td>
</tr>
<tr>
<td>large</td>
<td>20000</td>
<td>100,000,000</td>
<td>10020</td>
<td>10000</td>
<td>20000</td>
<td>10011</td>
</tr>
</tbody>
</table>

Table 5.2: Small, medium and larger scenarios results

If a new Endpoint is added to the network, assuming that after a change in the Endpoint’s list the complete filter is sent, we are in the same situation as when a new Participant is added to the network:

\[ N_{\text{marginalEndpoint}} = P \times ME \]  
\[ N_{\text{marginalEndpoint}} = P \times ME + 0.5 \times E/T \]

There is one message per Participant for announce itself and the BF:

\[ N_{\text{mmarginalParticipant}} = 1 + P \times ME \sim P \times ME \]
\[ N_{\text{mmarginalEndpoint}} = 1 + 0.5 \times E/T \]

If a new Endpoint is added to the network, the number of involved nodes (Participant) is the same:

\[ A_{\text{marginalEndpoint}} = P \]

Given the previous equations, some preliminary estimations can be provided for small, medium and larger scenarios. In particular we adopt the scenarios described in Table 5.1. The estimated evaluation, using previous analysis, is shown in Table 5.2.

5.4 Analytical study conclusions

The table and chart in Figure 5.3 have been done according to the analytical study equations for SDP and SDPBloom.
5.4. Analytical study conclusions

After a simple preliminary analysis, we show that our SPDBloom approach can reduce the resources requirements. The memory consumption and network messages are significantly smaller. However, note that the success of SPDBloom is conditioned by the particular network scenario. More precisely, the improvements are more significant as the number of Endpoints per Participant \( (E/P) \) increases. In the worst case, if \( E/P = 1 \), SPDBloom performance will be similar to SDP.
5. SPD with Bloom filters

Figure 5.3: SDPBloom analytical results
CHAPTER 6

Experimental results

For testing the real benefits of SDPBloom over SDP we have developed a testing tool called sdpb_tester. In this way we can validate the analytical results and combine both of them to provide a more complete evaluation.

The analytical study does not consider key composition used for discovery matching (the key composition for Bloom filters was introduced in Section 5.1). As mentioned in Section 2.1, there is a set of information that is sent in the discovery process (the Topic name, the type name, the type code) that can be suitable for key composition.

Since Bloom filters store keys as hash tables does, a priori we cannot insert on the filter non-keys attributes such as QoS parameters. The filters only allow membership queries but they do not support range queries, such as checking a range of a QoS attribute stored in the filter. For example, the filter can store a key such as “Radar 02” for supporting matching Topic name operations, but it is not valid for storing a QoS parameters, such as an offered deadline time. A remote node cannot check if a deadline period in the filter is smaller or bigger than its accepted deadline.

In they way Bloom filters work, the size of the bits array is independent of the key size. Hence, the more information we store in the filter, the more bandwidth is saved.

6.1 sdpb_tester tool

The sdpb_tester is implemented in C++ and it uses the Open Bloom Filter implementation by Arash Partow [47] and the Real-Time Innovations [48] DDS implementation version 4.3 [49] (formerly known as NDDS [50]). The purpose is to automatically create a set of different Topics and Endpoints in a node which must be matched in a remote node or set of nodes. The application extracts discovery information in the node, stores
6. Experimental results

it in a Bloom filter and publish it to the DDS data space. The remote node will receive
the Bloom filter and it will try to match its local Endpoints with the remote endpoints
by performing membership queries to the received filter. sdpb_tester reports the size
of the discovery information represented as simple text, as SDP does, and the size of the
same information represented as a Bloom filter so we can measure the compression ratio.

Allowed options:

Generic options:
- v [ --version ] print version string
- h [ --help ] produce help message
- n [ --no-values ] do not print configuration parameters values
- g [ --gnuplot arg ] Writes the statistics data into the given filename

Bloom filter parameters. The key composition represents Topics and Endpoints.
Options allowed are any combination of 'topic', 'type', 'typecode' such as 'topic typecode':
- f [ --fp-prob arg (=1) ] False positive probability
- k [ --keys-number arg (=1) ] A priori keys number that the filter will store.
- a [ --key-all ] Use all the data available for the key
- t [ --key-topic ] [=arg(=1)] Add the topic name to the key
- y [ --key-type ] [=arg(=1)] Add the type name to the key
- c [ --key-typecode ] [=arg(=1)] Add the typecode to the key
- d [ --disc-domain-id arg (=0) ] Choose the domain ID for the Discovery

Domain
- d [ --disc-topic-name arg (=SDPBloom) ] Name for the SDPBloom discovery Topic
- p [ --publish ] [=arg(=1)] Make the SDPBloom publish discovery info
- s [ --subscribe ] [=arg(=1)] Make the SDPBloom subscribe to discovery info
- l [ --sleep arg (=4) ] Sleep period for publishing discovery data
- l [ --loops arg (=4) ] Loops for publish / subscribe the 'sleep period'

Topic and Endpoints creation options:
- d [ --domain-id arg (=2) ] Choose the domain ID for the DomainParticipant
- s [ --simple-r [=arg(=1)] ] Create N of DataReaders for data type 'Simple'. Implicit value is 1.
- s [ --simple-w [=arg(=1)] ] Create N of DataWriters for data type 'Simple'. Implicit value is 1.
- s [ --complex-r [=arg(=1)] ] Create N of DataReaders for data type 'Complex'. Implicit value is 1.
- s [ --complex-w [=arg(=1)] ] Create N of DataWriters for data type 'Complex'. Implicit value is 1.

Figure 6.1: sdpb_tester command line options

sdpb_tester execution options are listed in Figure 6.1. sdpb_tester offers the
following functionality:

- It allows to perform as a SDPBloom discovery information publisher, a subscriber
  or both of them.

46
• To create a set of Endpoints of two different data types. The purpose of having different data types is to test the influence of the data type code in the filter key composition.

• It permits to parametrize the Bloom filter options:
  – False positives probability rate.
  – Keys number that will be stored in the filter.
  – Customize the key that will be inserted into the filter. This is, to allow any combination of the *Topic name*, the *type name* and the *type code* as the key.

• Collect local discovery information and store it on the filter.

• Check local Endpoints against a received filter.

`sdpb_tester` builds the keys by appending a set of strings for obtaining a unique key identifying an Endpoint. For example, the key:

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

represents a key where:

• “W” tells 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 type code.

A remote DataReader of the same topic and type just need to build the same key and check if the key is in the filter.

For example, running `sdpb_tester` with the following parameters:

```
$ sdpb_tester -p -t --simple-w=1 --complex-w=2 --gnuplot=pub-kt
```

has the following effect:

• `--simple-w=1 --complex-w=2` creates 1 DataWriter for *Simple* type and 2 DataWriters for *Complex* type. The first *Complex* DataWriter will use the Topic “Complex Type 0” and the second on the Topic “Complex Type 1”.

• `-p` tells the application to publish discovery information.

• `-t` tells the application to use the topic name as the filter key.

• `--gnuplot=pub-kt` outputs experiment data in GNUPlot readable format. The output file name will be `pub-kt.dat`. 
6. Experimental results

6.2 Types IDL description

In Listing 6.1 we provide the IDL (Interface Description Language) descriptions of the used types for the tests and the discovery Topics. The DiscoveryBloomFilter is used for distributing discovery information. The Simple and Complex types are the types used for the tests. The Complex has a longer typecode than the Simple type.

```c
struct DiscoveryBloomFilter {
    long ParticipantID;
    octet pbf[2024];
    short keys_number;
    float fp_prob;
};

struct Simple {
    string msg;
};

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

Listing 6.1: Types IDL description

6.3 Test set

A set of tests has been conducted by changing the composition of the key in the filter and the number of Endpoints of Simple and Complex types.

Note that the discovery information we use is just the information needed for Endpoints matching: the Topic name, the data type name and the type code. We do not consider QoS parameters and other data specified in the RTPS standard since we consider this information is needed for establishing connections once a query operation is satisfied, and it is not needed by the discovery protocol.

Figures in sections 6.3.1 and 6.3.2 show the results of several tests comparing SDP and SDPBloom. The legend of each figure shows the key composition and Endpoints types for each test. The x axis shows the number of Endpoints created in a Domain Participant while the y axis shows the size of the sent discovery data or the compression ratio. The data used for obtaining these charts have been obtained from the Tables in Section 6.3.3.
6.3. Test set

6.3.1 Key composition influence

This section aims to check the Bloom filter key composition influence in terms of discovery data size. This information represents the Participants internal database that allows the matching process between local and remote Endpoints. It also measures the consumed bandwidth by discovery information for announcing Participant’s entities. The SDP data still constant in all the charts since we use it as the reference framework.

All the Figures show three different test: the first one uses the Simple data type for Endpoints data type, the second test uses the Complex data type and the third one uses both of them.

Figure 6.2 shows SDP and SDP Bloom using the Topic name as the key for Endpoint matching. Since the Topic name is the same for the three test, obviously the three test has identical results for SDP and SDPBloom. Similar comments apply to experiments in Figure 6.3 where the data type name is added to the key. At this point, we should remember the assumption done in Section 5.4 the improvements are more significant as the number of Endpoints per Participant \((E/P)\) increases. This applies to the sent messages to the network in the analytical section and now it is also valid for data size.

The data type code is added to the key composition in experiments at Figure 6.4. Therefore the key is composed of the topic name, the data type name and the type code. Now the tests produce different results for different types. It is showed that the best data compression is, obviously, for the data type with larger typecode, this is, the Complex type.

---

Figure 6.2: SDP vs SDP Bloom key topic
6. Experimental results

Figure 6.3: SDP vs SDPBloom key topic+typename

Figure 6.4: SDP vs SDPBloom key topic+typename+typecode
6.3.2 Compression ratio for each type and key combination

The purpose of this section is to analyse the compression ratio of discovery information represented with Bloom filters. The first aim is to check the filter compression performance for different number of keys (Endpoints). The second is to study how the compression ratio is affected for each data type and key composition.

Figures 6.5, 6.6, 6.7 show the information compression ratio as the number of Endpoints grow. Before performing the experiments, we though the compression ratio would grow with larger number of Endpoints. Moreover the results show that the compression ratio quickly tends to stabilize and to get a good ratio. We can notice that compression ratio seems to decrease for more Endpoints number with no apparent reason. This is explained because of the Bloom filter implementation [47] behaviour and because of how we calculate the compression ratio. Studying the Bloom filter implementation that we have used, we realize that some parameters are not constant.

Related to Open Bloom filters, for building the filter two parameters are required: the desired false positive rate and the number of keys that a priori we think the filter is going to store. Then, an optimal parameters estimation is done for obtaining the suitable bits array’s dimension for the filter.

In addition, the compression ratio is worked out by dividing the information size we transfer in SDPBloom by the discovery information size without using the filter. The structure DiscoveryBloomFilter at Listing 6.1 represents the SDPBloom discovery information used, which includes the filter bits vector. Since the bits array length is not a fixed parameter, the total SDPBloom data size can varies depending on the estimation of this parameter done when building the filter. This explains the variations in the compression ratio value.
6. Experimental results

Figure 6.5:

Figure 6.6:
6.3. Test set

Figure 6.7:
6. Experimental results

6.3.3 Generated data

The following tables have been used for generating the previous charts. The last column, **Endpoints type**, shows the combinations of Endpoints created in each Participant. These Endpoints discovery information is stored on the filter for the test. For example, “SW25CW25” tells that there were 25 DataWriters (and Topics) of the **Simple** Type and 25 DataWriters (and Topics) of the **Complex** Type.

It can be noticed that the false positive probability column has no fix values. As explained in previous section, some filter parameters are estimated by the Open Bloom filter implementation while working with the filter. In the case of false positives rate, each time a key is added to the filter, the false positive probability is updated according to the bits array length and the total keys numbers stored in the filter. It makes sense since the false positive depends on the number of stored keys, as Equation (4.1) in Chapter 4 shows.

<table>
<thead>
<tr>
<th>Endpoints number</th>
<th>SDP Size</th>
<th>SDPBloom Size</th>
<th>Compression ratio</th>
<th>FP Probability</th>
<th>Endpoints type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>8</td>
<td>1.75</td>
<td>0.01</td>
<td>SW1</td>
</tr>
<tr>
<td>10</td>
<td>140</td>
<td>24</td>
<td>5.8333</td>
<td>0.001</td>
<td>SW10</td>
</tr>
<tr>
<td>25</td>
<td>365</td>
<td>57</td>
<td>6.4035</td>
<td>0.0004</td>
<td>SW25</td>
</tr>
<tr>
<td>50</td>
<td>740</td>
<td>117</td>
<td>6.3247</td>
<td>0.0002</td>
<td>SW50</td>
</tr>
<tr>
<td>100</td>
<td>1490</td>
<td>246</td>
<td>6.0569</td>
<td>0.0001</td>
<td>SW100</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>8</td>
<td>1.875</td>
<td>0.01</td>
<td>CW1</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
<td>24</td>
<td>6.25</td>
<td>0.001</td>
<td>CW10</td>
</tr>
<tr>
<td>25</td>
<td>375</td>
<td>57</td>
<td>6.5789</td>
<td>0.0004</td>
<td>CW25</td>
</tr>
<tr>
<td>50</td>
<td>750</td>
<td>117</td>
<td>6.4102</td>
<td>0.0002</td>
<td>CW50</td>
</tr>
<tr>
<td>100</td>
<td>1500</td>
<td>246</td>
<td>6.0975</td>
<td>0.0001</td>
<td>CW100</td>
</tr>
<tr>
<td>2</td>
<td>29</td>
<td>9</td>
<td>3.2222</td>
<td>0.005</td>
<td>SW1CW1</td>
</tr>
<tr>
<td>10</td>
<td>145</td>
<td>24</td>
<td>6.0417</td>
<td>0.001</td>
<td>SW5CW5</td>
</tr>
<tr>
<td>20</td>
<td>290</td>
<td>46</td>
<td>6.3043</td>
<td>0.0005</td>
<td>SW10CW10</td>
</tr>
<tr>
<td>50</td>
<td>740</td>
<td>117</td>
<td>6.3247</td>
<td>0.0002</td>
<td>SW25CW25</td>
</tr>
<tr>
<td>100</td>
<td>1490</td>
<td>246</td>
<td>6.0569</td>
<td>0.0001</td>
<td>SW50CW50</td>
</tr>
</tbody>
</table>

Table 6.1: SDP vs. SDPBloom with key topicname
### 6.3. Test set

#### Table 6.2: SDP vs. SDPBloom with key topicname+typename

<table>
<thead>
<tr>
<th>Endpoints number</th>
<th>SDP Msg Size</th>
<th>SDPBloom Msg Size</th>
<th>Compression ratio</th>
<th>FP Probability</th>
<th>Endpoints type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>8</td>
<td>2.3</td>
<td>0.01</td>
<td>SW1</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
<td>24</td>
<td>0.33333</td>
<td>0.001</td>
<td>SW10</td>
</tr>
<tr>
<td>25</td>
<td>515</td>
<td>57</td>
<td>9.03509</td>
<td>0.0004</td>
<td>SW25</td>
</tr>
<tr>
<td>50</td>
<td>1040</td>
<td>117</td>
<td>8.88889</td>
<td>0.0002</td>
<td>SW50</td>
</tr>
<tr>
<td>100</td>
<td>2090</td>
<td>246</td>
<td>8.49593</td>
<td>0.0001</td>
<td>SW100</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>8</td>
<td>2.75</td>
<td>0.01</td>
<td>CW1</td>
</tr>
<tr>
<td>10</td>
<td>220</td>
<td>24</td>
<td>9.16667</td>
<td>0.001</td>
<td>CW10</td>
</tr>
<tr>
<td>25</td>
<td>550</td>
<td>57</td>
<td>9.64912</td>
<td>0.0004</td>
<td>CW25</td>
</tr>
<tr>
<td>50</td>
<td>1100</td>
<td>117</td>
<td>9.40171</td>
<td>0.0002</td>
<td>CW50</td>
</tr>
<tr>
<td>100</td>
<td>2200</td>
<td>246</td>
<td>8.94309</td>
<td>0.0001</td>
<td>CW100</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>9</td>
<td>4.66667</td>
<td>0.005</td>
<td>SW1CW1</td>
</tr>
<tr>
<td>10</td>
<td>210</td>
<td>24</td>
<td>8.75</td>
<td>0.001</td>
<td>SW5CW5</td>
</tr>
<tr>
<td>20</td>
<td>420</td>
<td>46</td>
<td>9.13043</td>
<td>0.0005</td>
<td>SW10CW10</td>
</tr>
<tr>
<td>50</td>
<td>1065</td>
<td>117</td>
<td>9.10256</td>
<td>0.0002</td>
<td>SW25CW25</td>
</tr>
<tr>
<td>100</td>
<td>2140</td>
<td>246</td>
<td>8.69919</td>
<td>0.0001</td>
<td>SW50CW50</td>
</tr>
</tbody>
</table>

#### Table 6.3: SDP vs. SDPBloom with key topicname+typename+typecode

<table>
<thead>
<tr>
<th>Endpoints number</th>
<th>SDP Msg Size</th>
<th>SDPBloom Msg Size</th>
<th>Compression ratio</th>
<th>FP Probability</th>
<th>Endpoints type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53</td>
<td>8</td>
<td>6.625</td>
<td>0.01</td>
<td>SW1</td>
</tr>
<tr>
<td>10</td>
<td>530</td>
<td>24</td>
<td>22.0833</td>
<td>0.001</td>
<td>SW10</td>
</tr>
<tr>
<td>25</td>
<td>1340</td>
<td>57</td>
<td>23.5088</td>
<td>0.0004</td>
<td>SW25</td>
</tr>
<tr>
<td>50</td>
<td>2690</td>
<td>117</td>
<td>22.9915</td>
<td>0.0002</td>
<td>SW50</td>
</tr>
<tr>
<td>100</td>
<td>5390</td>
<td>246</td>
<td>21.9106</td>
<td>0.0001</td>
<td>SW100</td>
</tr>
<tr>
<td>1</td>
<td>132</td>
<td>8</td>
<td>16.5</td>
<td>0.01</td>
<td>CW1</td>
</tr>
<tr>
<td>10</td>
<td>1320</td>
<td>24</td>
<td>55</td>
<td>0.001</td>
<td>CW10</td>
</tr>
<tr>
<td>25</td>
<td>3300</td>
<td>57</td>
<td>57.8947</td>
<td>0.0004</td>
<td>CW25</td>
</tr>
<tr>
<td>50</td>
<td>6600</td>
<td>117</td>
<td>56.4103</td>
<td>0.0002</td>
<td>CW50</td>
</tr>
<tr>
<td>100</td>
<td>13200</td>
<td>246</td>
<td>53.6585</td>
<td>0.0001</td>
<td>CW100</td>
</tr>
<tr>
<td>2</td>
<td>185</td>
<td>9</td>
<td>20.5556</td>
<td>0.005</td>
<td>SW1CW1</td>
</tr>
<tr>
<td>10</td>
<td>925</td>
<td>24</td>
<td>38.5417</td>
<td>0.001</td>
<td>SW5CW5</td>
</tr>
<tr>
<td>20</td>
<td>1850</td>
<td>46</td>
<td>40.2174</td>
<td>0.0005</td>
<td>SW10CW10</td>
</tr>
<tr>
<td>50</td>
<td>4640</td>
<td>117</td>
<td>39.6581</td>
<td>0.0002</td>
<td>SW25CW25</td>
</tr>
<tr>
<td>100</td>
<td>9290</td>
<td>246</td>
<td>37.7642</td>
<td>0.0001</td>
<td>SW50CW50</td>
</tr>
</tbody>
</table>
6. Experimental results
At the time of writing this project, other related works has recently being published. It corroborates that the content announcements by using Bloom filters is a suitable approach for many systems related to pure peer-to-peer architectures, as described in [21].

7.1 Best scenarios for using SDPBloom

As mentioned before in Section 5.4, SDPBloom is more suitable for scenarios with high $E/P$ relationship. There are distributed applications which use many DDS Topics per Participant because of the variety of events or information kind they manage even for non-large scenarios. Also, SDPBloom improvements over SDP are better in a unicast scenario. Typically multicast traffic is limited in WAN environments.

DDS applications are being integrated in WAN environments hence a set of issues appears. The massive scalability, the lack of multicast support and the firewall and NAT traversal affect to the DDS discovery process and publish-subscribe communication. The DDS Router [53] aims to integrate DDS applications over WAN by offering Domain bridging, Topic bridging and even Data Type transformation. Typically, a DDS router will connect several networks with a set of DDS applications. Figure 7.1 shows a DDS multi-site distributed application integrated with the DDS Router. DDS router will have to distribute discovery information through all the connected networks. We think the router could specially be benefited of using Bloom filter for summarize and distribute a DDS network content.
7. Conclusions and discussion

7.2 Bloom filter limitations

Bloom filters present a limitation: it is not possible to list the content of a filter. DDS systems use to check DDS Entities presence in the network by analysing discovery traffic. This is usually done for debugging purposes or for checking the presence of some DDS entities in reliable systems. Using Bloom filters can make debugging process more difficult because checking the content of a filter implies to a priori know the items that are stored in the filter.

A solution to this issue can be to allow the presence of the current SEDP. In this way, the Endpoints discovery information could be retrieved on-demand, as done when a filter match occurs between two Participants. If the process is done by using network analysing tools, the solution can be to set up the DDS system for periodically publish entities information. Hence a network analysing program could read the Endpoint’s information traffic. This publish period does not need to be the same as the Participant announcements period and it could be controlled by a QoS policy such as other DDS features.

7.3 Conclusions

In this project we have presented a characterization of the current DDS discovery protocol, the Simple Discovery Protocol. This characterization is the base for measuring discovery improvements such as the proposed SDPBloom, but it could be used in future
A review of the main features of some of the most significant discovery protocols was done in Chapter 3. Hence, we have used one of the discovery technologies found in this review, the Bloom filters. As future work, other discovery technologies could be applied for designing DDS discovery solutions. Similarly, a Bloom filters introduction and review have been done in Chapter 4.

The SDPBloom approach has been proposed as discovery solution for DDS with significant improvements. The advantages of SDPBloom can be summarized as:

- The number of sent messages to the network for Endpoints advertisement keeps constant while the number Endpoints increases.
- The Participant’s Endpoints information interchange is reduced to the matched Endpoints information, which is significantly smaller than the total Endpoints number transfered in SDP.
- SDPBloom improvements are better in scenarios with high number of Endpoints per Participant. For example, a Bloom filter can be used for summarize a DDS network content in the DDS Router [53].
- The more information is added to the key, the best compression ratio is reached. The typecode specially increases the compression ratio.

In addition to the SDPBloom advantages, we should mention the SDPBloom’s limitations:

- Debugging process can be more difficult with SDPBloom. However, solutions to this issue are proposed in Section 7.2.
- There can be false positives in Endpoints matching when using SDPBloom. Although the proposed algorithm considers the existence of false positives. This consideration makes the algorithm non-deterministic and the execution time can be variable. For some real-time systems a fixed execution time and deterministic algorithms are demanded. Therefore, a study of this issue should be done as future work.

### 7.4 Publications

This work was partially presented at OMG’s *Real-time and Embedded Systems Workshop*:


An extended version of this work is being submitted to a journal.


<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>Shortcut for Bloom filter</td>
</tr>
<tr>
<td>Bloom filter</td>
<td>A space-efficient probabilistic data structure that is used to test whether</td>
</tr>
<tr>
<td></td>
<td>an element is a member of a set</td>
</tr>
<tr>
<td>built-in entities</td>
<td>A special set of Topics, DataReaders and DataWriters for advertising and</td>
</tr>
<tr>
<td></td>
<td>discovering Participants and Endpoints</td>
</tr>
<tr>
<td>DataReader</td>
<td>DataReaders notify an application that data are available</td>
</tr>
<tr>
<td>DataWriter</td>
<td>Applications use DataWriters to write data to the GDS of a domain through</td>
</tr>
<tr>
<td></td>
<td>Publisher</td>
</tr>
<tr>
<td>DCPS</td>
<td>Data-Centric Publish-Subscribe</td>
</tr>
<tr>
<td>DDS</td>
<td>Data Distribution Service</td>
</tr>
<tr>
<td>DHT</td>
<td>Distributed Hash table</td>
</tr>
<tr>
<td>Domain</td>
<td>A Domain is a DDS virtual network</td>
</tr>
<tr>
<td>EDP</td>
<td>Endpoint Discovery Protocol</td>
</tr>
<tr>
<td>Endpoints</td>
<td>Data Readers and/or Data Writers</td>
</tr>
<tr>
<td>GDS</td>
<td>Global Data Space</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>Participant</td>
<td>A Domain Participant (or simply Participant) represents the participation of</td>
</tr>
<tr>
<td></td>
<td>the application on a Domain</td>
</tr>
<tr>
<td>Participant DATA</td>
<td>Shortcut for SPDPdiscoveredParticipantData</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PDP</td>
<td>Participant Discovery Protocol</td>
</tr>
<tr>
<td>peer-to-peer</td>
<td>A peer-to-peer (P2P) distributed network architecture is composed of participants that make a portion of their resources available directly to their peers without intermediary network hosts or servers</td>
</tr>
<tr>
<td>Publisher</td>
<td>A Publisher is an object responsible for data distribution</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RTPS</td>
<td>Real-time Publish-Subscribe</td>
</tr>
<tr>
<td>SDP</td>
<td>Simple Discovery Protocol</td>
</tr>
<tr>
<td>SDPBloom</td>
<td>SDP variation which takes advantages of Bloom filters</td>
</tr>
<tr>
<td>SEDP</td>
<td>Endpoint Discovery Protocol</td>
</tr>
<tr>
<td>SPDP</td>
<td>Simple Participant Discovery Protocol</td>
</tr>
<tr>
<td>SPDPdiscoveredParticipantData</td>
<td>In SDP, a especial message which is periodically sent to known peers when a Domain-Participant is created or deleted</td>
</tr>
<tr>
<td>Subscriber</td>
<td>A Subscriber is an object responsible for receiving published data and making it available to the receiving application</td>
</tr>
<tr>
<td>Topic</td>
<td>A Topic identified by its unique name in the whole domain defines the type of the DDS data objects</td>
</tr>
<tr>
<td>typecode</td>
<td>A DDS object data type structure description</td>
</tr>
</tbody>
</table>