Content-based Filtering Discovery Protocol (CFDP): Scalable and Efficient OMG DDS Discovery Protocol

Kyoungho An, Aniruddha Gokhale, and Douglas Schmidt
Dept of EECS, Vanderbilt University, Nashville, TN 37212, USA
{kyoungho.an, a.gokhale, d.schmidt}@vanderbilt.edu

Sumant Tambe, Paul Pazandak, and Gerardo Pardo-Castellote
Real-Time Innovations, Sunnyvale, CA 94089, USA
{sumant, paul, gerardo}@rti.com

ABSTRACT

The OMG Data Distribution Service (DDS) has been deployed in many mission-critical systems and increasingly in Internet of Things (IoT) applications since it supports a loosely-coupled, data-centric publish/subscribe paradigm with a rich set of quality-of-service (QoS) policies. Effective data communication between publishers and subscribers requires dynamic and reliable discovery of publisher/subscriber endpoints in the system, which DDS currently supports via a standardized approach called the Simple Discovery Protocol (SDP). For large-scale systems, however, SDP scales poorly since the discovery completion time grows as the number of applications and endpoints increases. To scale to much larger systems, a more efficient discovery protocol is required.

This paper makes three contributions to overcoming the current limitations with DDS SDP. First, it describes the Content-based Filtering Discovery Protocol (CFDP), which is our new endpoint discovery mechanism that employs content-based filtering to conserve computing, memory and network resources used in the DDS discovery process. Second, it describes the design of a CFDP prototype implemented in a popular DDS implementation. Third, it analyzes the results of empirical studies conducted in a testbed we developed to evaluate the performance and resource usage of our CFDP approach compared with SDP.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

Keywords

Discovery, Data Distribution Service, Pub/Sub, P2P

1. INTRODUCTION

The publish/subscribe (pub/sub) communication paradigm [1] is attractive due to its inherent scalability, which decouples publishers and subscribers of event data in time and space, and enables them to remain anonymous and communicate asynchronously. A data subscriber is an entity that consumes data by registering its interest in a certain format (*e.g.*, topics, types or content) at any location and at any time. Likewise, a data publisher is an entity that produces data for consumption by interested subscribers.

A key requirement for the pub/sub paradigm is the discovery of publishers by subscribers. Although anonymity is often a key trait of pub/sub, the underlying pub/sub mechanisms that actually deliver the data from publishers to subscribers must map subscribers to publishers based on matching interests. To support spatio-temporal decoupling of publishers and subscribers, an efficient and scalable discovery mechanism is essential for pub/sub systems since publishers and subscribers need not be present at the same time and in the same location.

Achieving efficient and scalable discovery is even more important for pub/sub systems that require stringent quality-of-service (QoS) properties, such as latency of data delivery, reliable delivery of data, or availability of historical data. To meet the requirements of various systems, a range of discovery mechanisms exist. Example mechanisms include using static and predetermined lookups, using a centralized broker, or using a distributed and decentralized approach.

The OMG *Data Distribution Service* (DDS) [9, 11] is a standardized pub/sub middleware that provides a range of QoS properties to distributed real-time and embedded (DRE) systems in which discovery is a key challenge. Since DDS supports QoS policies that enable it to disseminate data in a reliable and real-time manner, it has been used to build many mission-critical DRE systems, such as air traffic control, unmanned vehicles, and industrial automation systems.

The discovery mechanism defined in the DDS standard is based on a peer-to-peer (P2P) protocol, where a peer automatically discovers other peers by matching the topic names, their data types, and their selected QoS configurations. DDS peers that contain the endpoints (*i.e.*, actual data publishers or subscribers) are required to locate remote matching peers with their endpoints to establish communication paths. Each peer thus runs a discovery protocol to find matching remote endpoints.

The DDS standard adopts a distributed and decentralized approach called the *Simple Discovery Protocol* (SDP) [10]. SDP provides simple and flexible system management by using discovery traffic for joining and leaving endpoints. It also supports updating the QoS status of endpoints.

Although SDP is standardized, one drawback is that it scales poorly as the number of peers and their endpoints increases in a domain since each peer sends/receives discovery messages to/from other peers in the same domain [16]. When a large-scale DRE system is deployed with SDP, therefore, substantial network, memory and computing resources are consumed by every peer just for the discovery process. This overhead can significantly degrade discovery completion time and hence the overall scalability of a DDS-based pub/sub system.

The root cause of SDP's scalability issues is that peers send discovery messages to every other peer in the domain, yet perhaps only a fraction of the peers are actually interested in conversing with any other peer. As a result, unnecessary network, computing and memory resources are used for this discovery protocol. For example, a data consumer receives and keeps discovery objects (these objects describe all of topic names and data type formats of the data that each consumer uses) for all other consumers even though they never communicate with each other since they are identical types of endpoints (*e.g.*, they are all subscribers).

To overcome this limitation with SDP, this paper presents a new mechanism for scalable DDS discovery called the *Content-based Filtering Discovery Protocol* (CFDP). CFDP employs content filtering on the sending peer(s) to filter discovery messages by exchanging filtering expressions that limit the range of interests *a priori*. To implement SDP, we created a special DDS topic called the *Content Filtered Topic (CFT)*, which includes filtering properties that are composed of a filtering expression and a set of parameters used by that expression. By using CFT, peers on the sending side that use CFDP can filter unwanted discovery messages and enhance the efficiency and scalability of the discovery process. The results of our empirical evaluations presented in Section 4 demonstrate a linear reduction in the number of transferred and stored messages.

The remainder of this paper is organized as follows: Section 2 summarizes the key elements of OMG DDS and SDP; Section 3 describes the design and implementation of CFDP; Section 4 analyzes the results of empirical evaluations of CFDP; Section 5 compares our CFDP approach with related work; and Section 6 presents concluding remarks.

2. AN OVERVIEW OF OMG DDS AND ITS SDP DISCOVERY SERVICE

This section summarizes the key elements of the OMG *Data Distribution Service* (DDS) and its *Simple Discovery Protocol* (SDP).

2.1 Overview of the OMG Data Distribution Service (DDS)

The OMG DDS specification defines a distributed pub-/sub communications architecture [9]. At the core of DDS is a data-centric architecture for connecting anonymous data publishers with data subscribers, as shown in Figure 1. The DDS architecture promotes loose coupling between system components. The data publishers and subscribers are decoupled with respect to (1) time (*i.e.*, they need not be present at the same time), (2) space (*i.e.*, they may be located anywhere), (3) flow (*i.e.*, data publishers must offer equivalent or better quality-of-service (QoS) than required by data subscribers), and behavior (*i.e.*, business logic-independent), (4) platforms, and (5) programming languages (*e.g.*, DDS applications can be written in many programming languages, including C, C++, Java, and Scala).

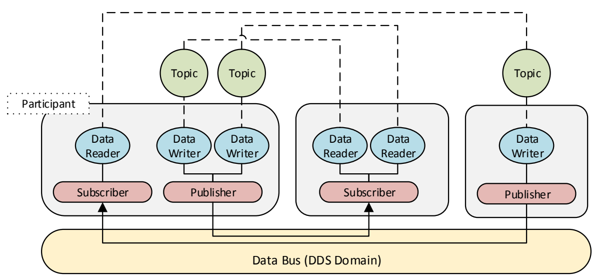


Figure 1: DDS Architecture

A DDS data publisher produces typed data-flows identified by names called *topics*. The coupling between a publisher and subscriber is expressed only in terms of topic name, its data type schema, and the offered and requested QoS attributes of publishers and subscribers, respectively. Below we briefly describe the key architectural elements of the DDS specification.

- **Domain** is a logical communication environment used to isolate and optimize network communications within the group of distributed applications that share common interests (*i.e.*, topics and QoS). DDS applications can send and receive data among themselves only if they have the same domain ID.

- **Participant** is an entity that represents either a publisher or subscriber role of a DDS application in a domain, and behaves as a container for other DDS entities (*i.e.*, DataWriters and DataReaders), which are explained next.

- **DataWriter and DataReader.** *DataWriters* (data publishers) and *DataReaders* (data subscribers) are endpoint entities used to write and read typed data messages from a global data space, respectively. DDS ensures that the endpoints are compatible with respect to topic name, their data type, and QoS configurations. Creating a DataReader with a known topic and data type implicitly creates a *subscription*, which may or may not match with a DataWriter depending upon the QoS.

- **Topic** is a logical channel between DataWriters and DataReaders that specifies the data type of publication and subscription. The topic names, types, and QoS of DataWriters and DataReaders must match to establish communications between them.

- **Quality-of-service (QoS).** DDS supports around two dozen QoS policies that can be combined in different ways. Most QoS policies have requested/offered semantics, which are used to configure the data flow between each pair of DataReader and DataWriter, and dictate the resource usage of the involved entities.

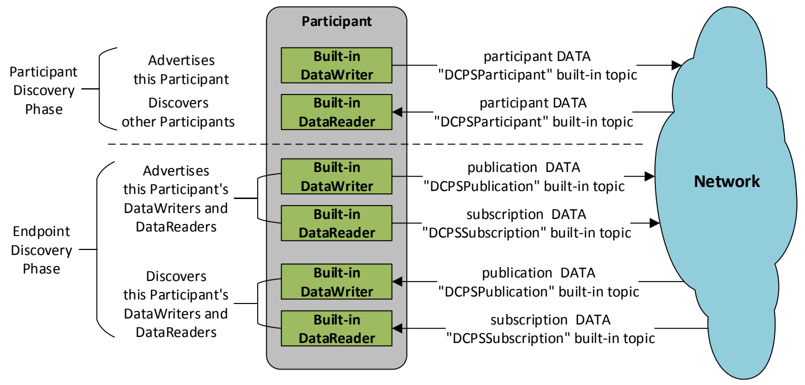

Participant
Participant Discovery Phase
Advertises this Participant
Discovers other Participants
Built-in DataWriter
Built-in DataReader
participant DATA "DCPSParticipant" built-in topic
participant DATA "DCPSParticipant" built-in topic
Endpoint Discovery Phase
Advertises this Participant's DataWriters and DataReaders
Discovers this Participant's DataWriters and DataReaders
publication DATA "DCPSPublication" built-in topic
subscription DATA "DCPSSubscription" built-in topic
publication DATA "DCPSPublication" built-in topic
subscription DATA "DCPSSubscription" built-in topic
Network

Figure 2: DDS Discovery Protocol Built-in Entities

- **Content Filtered Topic (CFT).** Content filters refine a topic subscription and help to eliminate samples that do not match the defined application-specified predicates. The predicate is a string encoded SQL-like expression based on the fields of the data type. The filter expression and the parameters may change at run-time. Data filtering can be performed by the DataWriter or DataReader.

2.2 Overview of the DDS Simple Discovery Protocol (SDP)

The OMG DDS *Real-Time Publish-Subscribe* (RTPS) standard [10] defines a discovery protocol that splits the discovery process into two phases: the *Participant Discovery Protocol* (PDP) and the *Endpoint Discovery Protocol* (EDP). The PDP defines the means for discovering participants in a network. After participants have discovered each other, they exchange discovery messages for endpoints via the EDP.

The standard DDS specification describes a concrete discovery protocol called the *Simple Discovery Protocol* (SDP) as the default discovery protocol to be used by different DDS implementations for interoperability. SDP also uses the two phase approach, which are called the *Simple Participant Discovery Protocol* (SPDP) and *Simple Endpoint Discovery Protocol* (SEDP). These discovery protocols are suitable for deployments of DDS in *Local-Area Networks* (LANs).

In the SPDP phase, a participant in a DDS application uses multicast or unicast discovery messages to announce named *participant DATA* to other participants periodically. These messages use built-in topics for discovery to let existing participants know of a new "announcing" participant. Figure 2 depicts the different built-in DDS entities (topics and endpoints) used by SDP. The SPDP message contains a participant's *Globally Unique Identifier* (GUID), transport locators (IP addresses and port numbers), and QoS policies. The message is periodically sent with the BEST_EFFORT reliability QoS to maintain liveliness of discovered participants. When *participant DATA* messages are received from other participants, the received messages for remote participants are archived in a database managed by the DDS middleware.

After a pair of remote participants discover each other by exchanging discovery messages, they transition to the SEDP phase. In this phase, remote entities (*i.e.*, remote participants or endpoints) imply the remotely-located entities from the entity that initiated the discovery service. Likewise, local entities (*i.e.*, local participants or endpoints) indicate the locally-located entities in the same process address space.

After the SPDP phase, SEDP begins to exchange discovery messages of endpoints using the RELIABLE reliability QoS. These messages are known as *publication DATA* for DataWriters and *subscription DATA* for DataReaders, respectively. They include topic names, data types, and QoS of discovered endpoints. In the SEDP phase, participants archive received remote endpoints' information into an internal database and start the matching process. During this process the communication paths for publication and subscription are established between the matching endpoints if the remote endpoints have the same topic name, data types, and compatible QoS configurations with the ones of their local endpoints.

Rather than using an out-of-band mechanism for discovery, SDP uses DDS entities as an underlying communication transport for exchanging discovery information outlined above. As an initial step, an SDP-enabled participant creates and uses a special type of DDS entity called a "built-in entity" to publish and subscribe discovery data. This built-in entity uses predefined built-in topics (*DCPSParticipant*, *DCPSPublication*, and *DCPSSubscription*) to discover remote participants and endpoints, as shown in Figure 2.

The built-in entity has two variants: DataWriters for announcement and DataReaders for discovery. Each discovery topic for different entities (participant, publication, and subscription) therefore has a pair of DataWriters and DataReaders. After this built-in entity is initially created, the participants use it to exchange discovery messages for SDP.

3. CFDP DESIGN AND IMPLEMENTATION

Section 1 highlighted the drawbacks of the existing standardized protocol for peer discovery in DDS called *Simple Discovery Protocol* (SDP). The key limitations in SDP stemmed from multicasting discovery messages to all other peers in the LAN, many of whom may not have a match between the publication and subscription. This protocol unnecessarily wastes network, storage and computation resources. To overcome this limitation, we designed the *Content-based Filtering Discovery Protocol* (CFDP). This section first describes the design of CFDP and then outlines key elements of its implementation.

3.1 The Design of CFDP

CFDP filters discovery messages based on topic names and endpoint types, thereby reducing the number of resources wasted by the traditional SDP approach. Like SDP, CFDP utilizes the first phase of SDP called SPDP (described in Section 2.2) for participant discovery. It differs from SDP in the endpoint discovery phase known as SEDP, where key modifications have been designed for CFDP.

Similar to how SEDP applies DDS built-in entities as a communication transport to share discovery information, CFDP also uses DDS built-in entities to exchange discovery messages, however with some modifications.

In our design, we have used a special feature called *Content Filtered Topic* (CFT) that is supported in our underlying DDS implementation. CFT filters data samples on the DataWriter or DataReader side in accordance with the filtering expression defined in a DataReader.

CFDP's key enhancement was to create built-in entities with CFTs that filter discovery messages on topic names stored in *subscription DATA* and *publication DATA*. Since built-in topics already exist for discovering publication and subscription in the SEDP design, the CFDP creates separate built-in CFTs and filtering expressions for DataWriters and DataReaders. The application logic is completely oblivious to these steps because everything is handled at the DDS middleware-level.

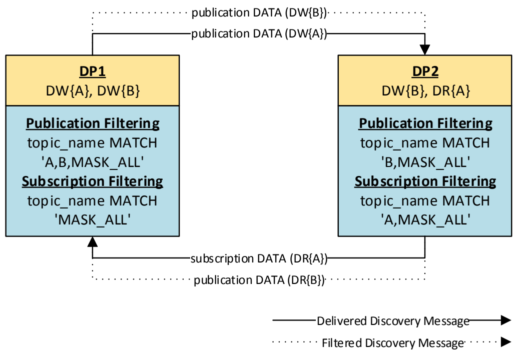


Figure 3: Filtering Discovery Messages by Topic Names

Figure 3 shows an example of filtering discovery messages based on topic names. The following notations are used in the figure.

- DPx - Domain Participant named x
- Discovery DB - Internal in-memory database stored at the core level of the DDS middleware.
- DW$\{y\}$ - DataWriter publishing a topic y
- DR$\{z\}$ - DataReader subscribing for a topic z
- DPx[DW$\{y\}$, DR$\{z\}$] - Discovery object indicating a Domain Participant containing DW$\{y\}$ and DR$\{z\}$ stored in the Discovery DB.

DPs using SDP disseminate all the discovery messages for created endpoints to other DPs in the same domain. Our CFDP approach, however, filters out unmatched discovery messages on the DataWriter side utilizing CFTs by harnessing topic names of local endpoints. This avoids unwanted messages being sent over the network.

In this example, *DP1* does not forward any discovery messages about $DW\{B\}$ to *DP2* since *DP2* does not have $DR\{B\}$. Likewise, *DP2* does not send discovery messages for $DW\{B\}$ to *DP1*. Discovery messages about unmatched endpoints in CFDP are filtered out on the announcing side and implemented via filtering expressions defined in each participant.

The publication filtering expression in *DP1* (*topic_name MATCH 'A,B,MASK_ALL'*) means that it only accepts subscription discovery messages for topics A and B. Here, *topic_name* is a variable defined in the built-in topic for the discovery process. *MATCH* is a reserved relational operator for a CFT expression. If a value of the variable is matched with the right-hand operator (*'A,B,MASK_ALL'*), a sample containing the value is accepted by the participant. Likewise, the subscription filtering expression does not allow any publication discovery messages by using a reserved filter, *MASK_ALL*, because DataReaders do not exist.

Figures 4 and 5 compare SDP and CFDP by showing an example having the same application topology. In this comparison, we assume that unicast is used for the discovery protocol. In Figure 4, each participant contains DataWriters and DataReaders with topic names ranging from A to D. Every participant publishes SEDP messages of endpoints to other participants in the same domain, and participants receiving the discovery messages store the messages as discovery objects in internal database. Six discovery objects including discovery objects of local endpoints are stored in each participant resulting in a total of 18 discovery objects consuming memory resources in this system. Likewise, there are a total of 18 network transfers.

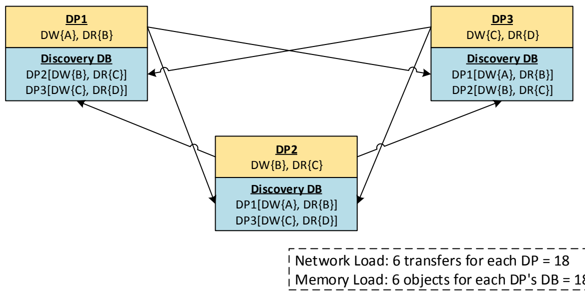


Figure 4: SDP Example

Figure 5 shows a case using CFDP, which has the same setup and topology used in the SDP example. However, in this case, each participant filters discovery messages based on topic names and endpoint types, so it transfers and stores only the required discovery messages. As a result, a total of ten discovery objects are stored in each local database resulting in ten network transfers. This comparison demonstrates that CFDP can conserve memory and network resources.

Algorithm 1 describes the pseudo code for the event callback functions for CFDP. The callback function is invoked by the pluggable discovery framework described in Section 3.3.1, which we have used in our solution.

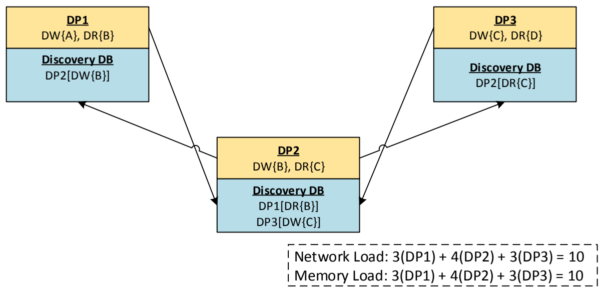


Figure 5: CFDP Example

Algorithm 1 CFDP Callback Function Algorithms

```
function LOCAL ENDPOINT ENABLED($EP$)
    if $EP_{type}$ is $DataWriter$ then
        if $EP_{topicName} \in SubFiltering$ then
            Add $EP_{topicName}$ to $SubFiltering$
    else if $EP_{type}$ is $DataReader$ then
        if $EP_{topicName} \in PubFiltering$ then
            Add $EP_{topicName}$ to $PubFiltering$
    Send EP to remote Participants
function LOCAL ENDPOINT DELETED($EP$)
    if $EP_{type}$ is $DataWriter$ then
        if $EP_{topicName} \in SubFiltering$ then
            Delete $EP_{topicName}$ from $SubFiltering$
    else if $EP_{type}$ is $DataReader$ then
        if $EP_{topicName} \in PubFiltering$ then
            Delete $EP_{topicName}$ from $PubFiltering$
function REMOTE DATA WRITER RECEIVED($EP_{DW}$)
    Assert $EP_{DW}$ into Internal DB
function REMOTE DATA READER RECEIVED($EP_{DR}$)
    Assert $EP_{DR}$ into Internal DB
```

Each callback function is invoked when the following events occur:

- *LocalEndpointEnabled* - when a local endpoint (DataWriter or DataReader) is created
- *LocalEndpointDeleted* - when a local endpoint (DataWriter or DataReader) is deleted
- *RemoteDataWriterReceived* - when a remote DataWriter is created
- *RemoteDataReaderReceived* - when a remote DataReader is created

A discovery object of a created local endpoint is delivered as a parameter value to the *LocalEndpointEnabled* callback function. When the discovery object is for a local DataWriter, the function adds the DataWriter's topic name to the subscription filtering expression(*SubFiltering*). The subscription filtering expression is used for finding matching remote DataReaders for local DataWriters.

Similarly, in the case for finding matching remote DataWriters for local DataReaders, the publication filtering expression (*PubFiltering*) is used and must be updated with a topic name of a local DataReader if it does not exist. After updating topic names in the filtering expressions, it sends the discovery objects to other participants to let them know about the newly created local endpoints.

When a local endpoint is deleted, the topic name of the local endpoint is removed from the relevant filtering expressions in the callback function *LocalEndpointDeleted*. The callback functions for remote endpoints (*RemoteDataWriterReceived* and *RemoteDataReaderReceived*) insert discovery objects of remote endpoints to the internal database. The underlying DDS middleware then executes the matching process by comparing data types and QoS policies stored in the discovery objects to establish communication paths between endpoints.

3.2 Analysis of SDP and CFDP Complexity

We performed a complexity analysis of SDP and CFDP to determine the expected number of transmitted discovery messages and stored discovery objects. Tables 1 and 2 show notations and metrics used for the complexity analysis, respectively.

Table 1: Notation used for Complexity Analysis

Notation	Definition
P	Number of participants in a domain
E	Number of endpoints in a domain
F	Number of endpoints per participant, $Fanout = E/P$
R	Ratio of the number of matching endpoints to the number of endpoints in a participant, $0 \leq R \leq 1$

Table 2: Metrics used for Complexity Analysis

Metric	Definition
$N_{multi\_participant}$	Number of messages sent/received by a participant using multicast
N_{multi_total}	Total number of messages sent/received by participants using multicast in a domain
$N_{uni\_participant}$	Number of messages sent/received by a participant using unicast
N_{uni_total}	Total number of messages send/received by participants using unicast in a domain
$M_{participant}$	Number of endpoint discovery objects stored in a participant
M_{total}	Total number of endpoint discovery objects stored in a domain

The notations use the number of participants and the number of endpoints in a domain because these variables are crucial to measuring the performance of discovery protocols. A ratio of matching endpoints in each participant can also be used for CFDP complexity analysis because it is the primary factor affecting network transfers and stored objects of CFDP. For analysis metrics, we decided to inspect the number of transferred and stored discovery messages per participant, as well as per domain (entire set of participants) for both unicast and multicast.

3.2.1 SDP Complexity Analysis

In the case of multicast-enabled SDP, the number of messages sent from a participant is the number of endpoints divided by participants, which yields the number of endpoints in a participant, E/P. Regardless of the number of participants and endpoints in a domain, if multicast is used, a message is sent to other participants only when a local endpoint is created. The delivery of a message to multiple destinations (one-to-many communication) is performed by a network switch.

The number of messages received by a participant is the number of endpoints except for local endpoints in a participant ($\frac{E}{P}$ in Equation (1)). The number of messages sent from a participant is the number of local endpoints multiplied by the number of remote participants ($\frac{E}{P} \cdot (P-1)$ in Equation (1)). As a result, in Equation (2), $\frac{E}{P}$ is cancelled out and therefore the sum of sent and received number of messages per participant can be simplified to the number of endpoints in a domain, E in Equation (3). The total number of transfers in a domain is the number of transfers per participant multiplied by the number of participants in a domain, which is $E \cdot P$ as shown in Equation (4).

$$N_{multi\_participant} = \frac{E}{P} + \frac{E}{P} \cdot (P-1) \tag{1}$$

$$= \frac{E}{P} + E - \frac{E}{P} \tag{2}$$

$$= E \tag{3}$$

$$N_{multi\_total} = E \cdot P \tag{4}$$

Unicast-enabled SDP incurs more overhead when it sends discovery messages because it handles one-to-many data distribution by itself, rather than using network switches. Each participant disseminates the discovery message as many times as the number of endpoints without including local endpoints ($\frac{E}{P} \cdot (P-1)$ in Equation (5)). The number of received messages is the same as when using multicast ($\frac{E}{P} \cdot (P-1)$ in Equation (5)). The total number of network transfers incurred by the unicast enabled SDP is 2 times $\frac{E}{P} \cdot (P-1)$, and therefore $2 \cdot \frac{E}{P} \cdot (P-1)$, as shown in Equation (6).

$$N_{uni\_participant} = \frac{E}{P} \cdot (P-1) + \frac{E}{P} \cdot (P-1) \tag{5}$$

$$= 2 \cdot \frac{E}{P} \cdot (P-1) \tag{6}$$

$$\because (P-1) \sim P \tag{7}$$

$$\sim 2 \cdot E \tag{8}$$

$$N_{uni\_total} \sim 2 \cdot E \cdot P \tag{9}$$

In the asymptotic limit (i.e., very large P), as shown in Equation (7), $(P-1) \sim P$, and hence it can be approximated as $2 \cdot E$ in Equation (8), and accordingly the total number of transfers in a domain is roughly $2 \cdot E \cdot P$ as shown in Equation (9).

Unicast-enabled SDP incurs more overhead (which grows as a factor of E) compared to multicast-enabled SDP since every participant in a domain must send more messages since it uses point-to-point unicast data dissemination instead of the one-to-many multicast dissemination. SDP keeps all discovery objects of endpoints in the same domain, and therefore the memory consumption per participant for SDP is directly tied to the number of endpoints in a domain, E in Equation (10). The total memory capacity used by all participants in a domain is thus $E \cdot P$, as shown in Equation (11).

$$M_{participant} = E \tag{10}$$

$$M_{total} = E \cdot P \tag{11}$$

3.2.2 CFDP Complexity Analysis

We applied the same analysis as SDP to measure the number of transfers for CFDP, *i.e.*, a sum of received and sent network transfers. There is no change in the number of sent messages because each participant with multicast sends one message for each corresponding endpoint it contains ($\frac{E}{P}$ in Equation (12)). We can reduce the number of received messages, however, since only matched discovery messages are received by the filtering mechanism. This factor is therefore multiplied by the ratio of matching endpoints, R (for all, $0 \leq R \leq 1$). Hence, the number of received messages can be $\frac{E}{P} \cdot (P-1) \cdot R$ as shown in Equation (12). From Equation (12) to Equation (13), $\frac{E}{P} \cdot (P-1) \cdot R$ can be transitioned to $E \cdot R - \frac{E}{P} \cdot R$. Then, $\frac{E}{P} - \frac{E}{P} \cdot R$ in Equation (13) can be $\frac{E}{P} \cdot (1-R)$, and thus simply be $F \cdot (1-R)$ in Equation (14) because $\frac{E}{P} = F$.

$$N_{multi\_participant} = \frac{E}{P} + \frac{E}{P} \cdot (P-1) \cdot R \tag{12}$$

$$= \frac{E}{P} + E \cdot R - \frac{E}{P} \cdot R \tag{13}$$

$$= F \cdot (1-R) + E \cdot R \tag{14}$$

$$\sim E \cdot R \tag{15}$$

$$N_{multi\_total} \sim E \cdot P \cdot R \tag{16}$$

As shown above, the number of transfers per participant is approximated by $E \cdot R$ in Equation (15) because $F \cdot (1-R)$ in Equation (14) can be a very small number in most cases such that it can be ignored. Using the total transfers per participant ($E \cdot R$ in Equation (15)), the total transfers in a domain is $E \cdot P \cdot R$ in Equation (16).

CFDP reduces both the number of sent and received transfers proportional to the matching ratio if unicast is enabled, so both the number of sent and received messages can be $\frac{E}{P} \cdot (P-1) \cdot R$ in Equation (17). Accordingly, the number of transfers per participant is $2 \cdot \frac{E}{P} \cdot (P-1) \cdot R$ as shown in Equation (18). This number can be approximated to $2 \cdot E \cdot R$ shown in Equation (19) via the same analysis used for SDP (See Equation (7)). The total transfers in a domain is thus $2 \cdot E \cdot P \cdot R$, as shown in Equation (20) because it is the total number of transfers per participant ($2 \cdot E \cdot R$) multiplied by the number of participants in a domain (P).

$$N_{uni\_participant} = \frac{E}{P} \cdot (P-1) \cdot R + \frac{E}{P} \cdot (P-1) \cdot R \tag{17}$$

$$= 2 \cdot \frac{E}{P} \cdot (P-1) \cdot R \tag{18}$$

$$\sim 2 \cdot E \cdot R \tag{19}$$

$$N_{uni\_total} \sim 2 \cdot E \cdot P \cdot R \tag{20}$$

CFDP also decreases memory use as a consequence of reducing the number of remote endpoints by removing unmatched ones. Similarly, CFDP conserves resources proportional to the ratio of matching endpoints for cases per participant as shown in Equation (22), as well as per domain as shown in Equation (23).

$$M_{participant} = \frac{E}{P} + \frac{E}{P} \cdot (P-1) \cdot R \tag{21}$$

$$\sim E \cdot R \tag{22}$$

$$M_{total} \sim E \cdot P \cdot R \tag{23}$$

3.3 Implementing CFDP

We prototyped our solution at the application level, rather than make invasive and non-standard changes to the underlying DDS middleware, which is RTI Connext [14]. In particular, our implementation leveraged DDS discovery event callbacks using interfaces provided by the middleware, though our approach could be incorporated inside the middleware itself to optimize performance. Our prototype implementation assumes that when participants are created, users determine which topics are published or subscribed by participants (this decision is usually made when endpoints are created).

We made this assumption in our prototype implementation of CFDP since it uses the DDS TRANSIENT_LOCAL durability QoS for both late joiners and CFTs. When a parameter in a filtering expression of a CFT is changed, however, the changed value is not reflected in the list of late joiners. For example, a peer filters a discovery message of topic x because it does not have any endpoints interested in the topic x. Later when an endpoint interested in the topic x is created in the peer, then the peer should be considered a late joiner, but it is not. If the discovery message for the topic x is already filtered, it is not resent even though the filtering expression is updated with having the interest of topic x.

In addition, the prototype implementation of our CFDP discovery plugin had to address the following two issues:

1. It needs to operate with the DDS middleware core by interchanging discovery events since event information is available on different threads. Section 3.3.1 describes how we resolved this issue via the Pluggable Discovery Framework (PDF) provided by RTI Connext DDS [14].

2. It needs to have a proper data model according to the DDS RTPS specification for discovery messages exchanged between remote peers. Section 3.3.2 describes the data model we devised to address this issue.

3.3.1 *CFDP Plugin Using the Pluggable Discovery Framework (PDF)*

The PDF allows DDS users to develop pluggable discovery protocols, and utilize different discovery approaches under various system environments, such as a network environment with limited bandwidth or high loss rates. The PDF offers callback functions invoked when DDS entities are created, deleted, or changed. It also provides a function to assert a discovery object to internal database of the DDS middleware to delegate managing discovery objects and the matching process to the core level of the middleware. We therefore provided interfaces in PDF between the discovery plugins and associated participants as channels to exchange information in both directions: *local-to-remote* (announcing local entities) and *remote-to-local* (discovering remote entities).

Figure 6 shows the software architecture of our CFDP prototype. In this figure, there are six built-in entities present at the DDS core level. The CFDP uses SDP built-in entities to discover application-level built-in entities. The discovery plugin employs callback functions provided by the PDF to retrieve discovery events from the middleware.

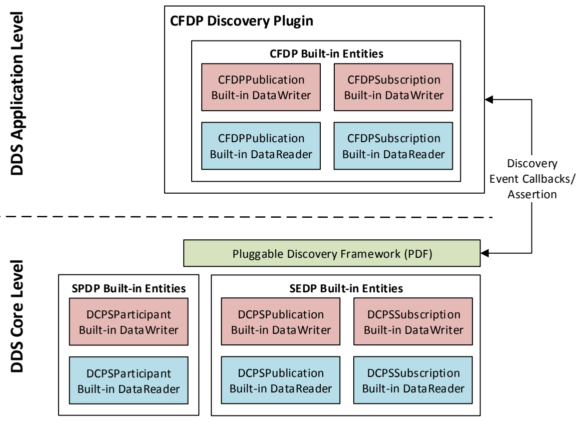

DDS Application Level
CFDP Discovery Plugin
CFDP Built-in Entities
CFDPPublication Built-in DataWriter
CFDPSubscription Built-in DataWriter
CFDPPublication Built-in DataReader
CFDPSubscription Built-in DataReader
Discovery Event Callbacks/ Assertion
Pluggable Discovery Framework (PDF)
DDS Core Level
SPDP Built-in Entities
DCPSParticipant Built-in DataWriter
DCPSParticipant Built-in DataReader
SEDP Built-in Entities
DCPSPublication Built-in DataWriter
DCPSSubscription Built-in DataWriter
DCPSPublication Built-in DataReader
DCPSSubscription Built-in DataReader

Figure 6: CFDP Prototype Software Architecture

Figure 7 introduces a procedure of discovery with the PDF enabled CFDP when a DataWriter is created.

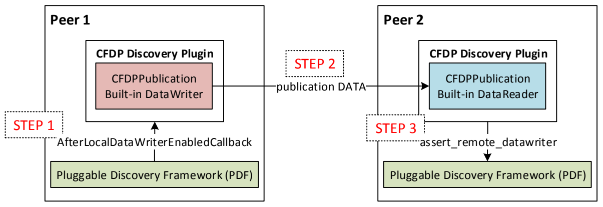
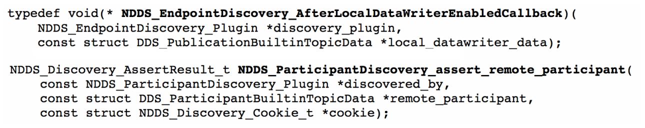

Peer 1
CFDP Discovery Plugin
CFDPPublication Built-in DataWriter
STEP 1
AfterLocalDataWriterEnabledCallback
Pluggable Discovery Framework (PDF)
STEP 2
publication DATA
Peer 2
CFDP Discovery Plugin
CFDPPublication Built-in DataReader
STEP 3
assert_remote_datawriter
Pluggable Discovery Framework (PDF)

Figure 7: CFDP Sequence with PDF

The PDF also furnishes a function to deliver discovery information from the application level to the core level of the middleware. The CFDP exchanges discovery messages between peers with four built-in entities like SDP, and sets up the same QoS configurations of SDP. For example, to deliver discovery messages in a reliable way, the RELIABLE reliability QoS is selected and the TRANSIENT_LOCAL durability QoS is configured to guarantee durability of discovery messages for late joiners as endpoints usually are not created at the same time.

Interfaces of functions for DataWriter discovery provided by the PDF are shown in Figure 8. The *NDDS_Endpoint*



```
typedef void(* NDDS_EndpointDiscovery_AfterLocalDataWriterEnabledCallback)(
    NDDS_EndpointDiscovery_Plugin *discovery_plugin,
    const struct DDS_PublicationBuiltinTopicData *local_datawriter_data);

NDDS_Discovery_AssertResult_t NDDS_ParticipantDiscovery_assert_remote_participant(
    const NDDS_ParticipantDiscovery_Plugin *discovered_by,
    const struct DDS_ParticipantBuiltinTopicData *remote_participant,
    const struct NDDS_Discovery_Cookie_t *cookie);
```

Figure 8: PDF Functions for DataWriter Discovery

Discovery_AfterLocalDataWriterEnabledCallback is a callback function invoked when a local DataWriter is created. CFDP uses this function to add a topic name of a created DataWriter to a proper filtering expression and to publish discovery information of the created DataWriter to announce to other remote peers. *NDDS_EndpointDiscovery _assert_remote_datawriter* is a function that asserts a discovery object of a discovered remote DataWriter to the internal database in the underlying middleware.

For example, if a DataWriter is produced in *Peer1* the *AfterLocalDataWriterEnabledCallback* is invoked and the callback function sends *Publication DATA* (discovery data for DataWriters) to other peers. The *publication DATA* then arrives at *Peer2* and is delivered to the core by calling the function *assert_remote_datawriter*. Finally, the DDS middleware establishes the communication by matching topic, type, and QoS policies stored in the *publication DATA*.

3.3.2 CFDP Data Model

A data model containing the required information for endpoints is needed to develop discovery built-in entities at the application level. DDS supports diverse QoS configurations and makes the data model for endpoint discovery complicated because QoS configurations for endpoints are exchanged at the endpoint discovery phase. Thus, defining the data model from scratch is hard. We therefore exploited OMG *Interface Definition Language* (IDL) definitions already used in the DDS core middleware and generated source code for the data model, thereby reducing the time and effort needed to define and implement the data model.

Figures 9 and 10 depict data models for publication and subscription discovery defined in IDL. Attributes for keys to

```
struct PublicationBuiltinTopicData {
  BuiltinTopicKey_t                     key;//@key
  BuiltinTopicKey_t                     participant_key;
  BuiltinTopicKey_t                     publisher_key;
  string                                topic_name;
  string                                type_name;

  DurabilityQosPolicy                   durability;
  DurabilityServiceQosPolicy            durability_service;
  DeadlineQosPolicy                     deadline;
  LatencyBudgetQosPolicy                latency_budget;
  LivelinessQosPolicy                   liveliness;
  ReliabilityQosPolicy                  reliability;
  LifespanQosPolicy                     lifespan;
  UserDataQosPolicy                     user_data;
  OwnershipQosPolicy                    ownership;
  OwnershipStrengthQosPolicy            ownership_strength;
  DestinationOrderQosPolicy             destination_order;

  PresentationQosPolicy                 presentation;
  PartitionQosPolicy                    partition;
  TopicDataQosPolicy                    topic_data;
  GroupDataQosPolicy                    group_data;
  TypeConsistencyEnforcementQosPolicy   type_consistency;
};
```

Figure 9: IDL Definition for Publication Discovery Data Model

identify discovery entities are contained and *topic_name* and *type_name* attributes can be used to find matching endpoints by topics and type structures. The data models include the basic attributes required for the DDS specification, but additional attributes can be added to support an advanced discovery process. For example, the type name can be used for type matching, but an object for type structure can be used to realize extensible and compatible type definitions.

QoS policies required for endpoint matching are also defined in the data models. The QoS policies are different depending on the types of entities (publication or subscription). For example, the time-based filter QoS controls data arrival rates on the DataReader side even though DataWriters may publish at a faster rate. This QoS policy can there-

```
struct SubscriptionBuiltinTopicData {
  BuiltinTopicKey_t                     key;//@key
  BuiltinTopicKey_t                     participant_key;
  BuiltinTopicKey_t                     subscriber_key;
  string                                topic_name;
  string                                type_name;

  DurabilityQosPolicy                   durability;
  DeadlineQosPolicy                     deadline;
  LatencyBudgetQosPolicy                latency_budget;
  LivelinessQosPolicy                   liveliness;
  ReliabilityQosPolicy                  reliability;
  OwnershipQosPolicy                    ownership;
  DestinationOrderQosPolicy             destination_order;
  UserDataQosPolicy                     user_data;
  TimeBasedFilterQosPolicy              time_based_filter;

  PresentationQosPolicy                 presentation;
  PartitionQosPolicy                    partition;
  TopicDataQosPolicy                    topic_data;
  GroupDataQosPolicy                    group_data;
  TypeConsistencyEnforcementQosPolicy   type_consistency;
};
```

Figure 10: IDL Definition for Subscription Discovery Data Model

fore be applied only to DataReaders (subscription). Likewise, the lifespan QoS validates how long samples are alive on the DataWriter side, and is only applicable to DataWriters (publication). DDS QoS policies can be reconfigured at run-time, and endpoint discovery messages are used to propagate those changes.

4. EMPIRICAL EVALUATION OF CFDP AND SDP

This section presents the results of empirical tests we conducted to compare CFDP with SDP. We conducted these tests to evaluate the scalability and efficiency of CFDP over SDP in terms of discovery completion time, which is defined as the duration of the discovery process to locate every matching endpoint in a domain. We measured CPU, memory, and network usage for both CFDP and SDP to determine how discovery completion time is affected by computation and network resource usage.

4.1 Overview of the Hardware and Software Testbed

Our testbed consists of six 12-core machines. Each machine has a 1 GB Ethernet connected to a single network switch. We implemented our CFDP plugin and the standard SDP implementation with RTI Connext DDS 5.0 [14]. We concurrently created 480 applications for each test, where each application contained a single participant. Specifically, we maintain equal number of publishers and subscribers, each with 20 endpoints (*i.e.*, a data writer or reader, respectively).

All test applications are evenly distributed and executed across the six machines. Each participant has 20 endpoints with an identical entity kind (only DataWriters or DataReaders), and the total number of endpoints in a test is 9,600 (*i.e.*, 480 applications with 20 endpoints each). We set the default matching ratio in each test to 0.1 (10%). It means two out of 20 endpoints in each participant are matched with other endpoints in a domain at a probability of 0.1. Our experiments have tested other matching ratios also.

CFDP uses unicast to filter messages on the DataWriter side because the filtering occurs on DataReader side if multicast is enabled. Filtering on the DataReader side filtering does not reduce the number of discovery messages transferred over a network, so it may have little improvement on performance. SDP uses multicast because it is the default discovery transport for this protocol.

We also developed a test application that measures discovery completion time. The application starts a timer after synchronizing distributed tests in a domain since we remotely execute the applications from a single machine sequentially. It then stops the timer when the expected number of endpoints are discovered. This implementation does not count the discovery time incurred for discovering participants, but measures only the time to discover endpoints because we compare the performance of the endpoint discovery phase (recall that the first phase of the discovery process is common to both SDP and CFDP).

4.2 Measuring Discovery Time

Discovery completion time is a critical metric to measure performance of discovery protocols. In the context of our tests, discovery completion time is defined as the time needed to discover all endpoints in a domain. We measured discovery completion time of 480 test applications for CFDP and SDP, respectively. Figure 11 presents the minimum, average, and maximum of discovery completion times for the test applications.

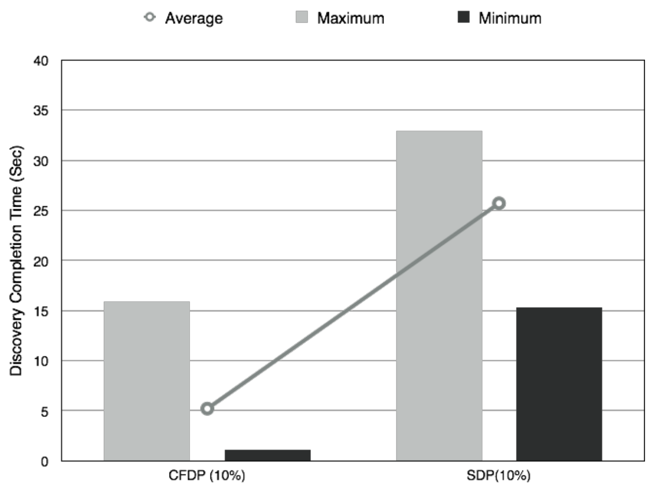


Figure 11: CFDP and SDP Discovery Time Comparison

The minimum discovery time for CFDP is 1.1 seconds and the maximum discovery is 15.92 seconds. Most test applications for CFDP complete the discovery process within 1 to 2 seconds. The average discovery time for all applications is 5.2 seconds. Some of the worst-case results were caused due to CPU saturation as discussed below.

In the case of SDP, the earliest discovery completion time is 15.3 seconds and the latest finish time is 32.9 seconds. The average discovery time is 15.3 seconds. As a result, the maximum discovery time of CFDP is 2 times faster than SDP; the average time is about 5 times faster, and the minimum discovery completion time is 15 times faster.

As mentioned above, the matching ratio used in this experiment is 0.1 (10%). In fact, SDP does not filter any discovery messages. So the different matching ratios make no difference for SDP. The performance of CFDP can be different based on the matching ratio, however, because it is determined by the number of messages to be filtered and its computational complexity.

If CFDP uses unicast as a transport, its performance can sometimes be worse than SDP with a higher matching ratio because it consumes computation resources for the filtering process as well as delivering messages which requires serializing and deserializing messages. CFDP therefore cannot always outperform SDP in a system with a smaller number of topics and endpoints. On the other hand, in large-scale DRE systems with many topics and endpoints, CFDP can always perform the discovery process more efficiently and scalably than SDP. Naturally, if the matching ratios are very high, then the benefit accrued may not be significant but we surmise that in general the matching ratios will remain small.

4.3 Measuring Resource Usage

We measured CPU utilization used by the discovery process by exploiting *gnome-system-monitor*. All distributed test applications are executed nearly simultaneously and the discovery process of each test application begins immediately after the endpoints are created. As shown in the Figure 12, however, there is a spike at a specific time range that occurs due to the discovery process overhead.

Figure 12 shows that more CPU cycles of SDP are consumed than for CFDP (10%). SDP's CPU utilization is higher since the number of transferred discovery messages is larger than for CFDP, so it incurs more processing overhead. These results indicate that the processing overhead incurred by filtering messages on DataWriters is lower than the processing costs incurred by transferring discovery messages. We therefore conclude that the CFDP discovery scheme uses fewer CPU resources than SDP.

The processing resources used by CFDP can be different for different matching ratios, as shown by CPU utilization results in the Figure 12. As expected, CPU utilization increases with higher matching ratios. This figure indicates that CFDP is not effective for every case, especially with a high matching ratio. It should therefore be used for large-scale DRE system having many endpoints where the matching ratio is lower than a small-scale system.

To analyze and compare network and memory usage of CFDP and SDP, we counted the number of sent and received messages for the discovery process by each protocol. The DataWriters and DataReaders provide detailed messaging information, such as how many messages and bytes are sent, received, or filtered via the functions named *get_datawriter_protocol_status()* and *get_datareader_protocol_status()*. Based on this information, we counted the number of messages, as shown in Figure 13.

CFDP exchanges discovery messages via unicast and each test application creates the same number of endpoints. The number of received and sent messages are therefore identical. CFDP (10%) sent and received 480 samples each and the total was 960.

SDP uses multicast for discovery messages, so the number of received messages is substantially more than the number of sent messages. In this experiment, only 20 samples are sent by a participant as each participant creates 20 endpoints. Although SDP sends fewer messages than CFDP, the total number of messages transferred by a participant

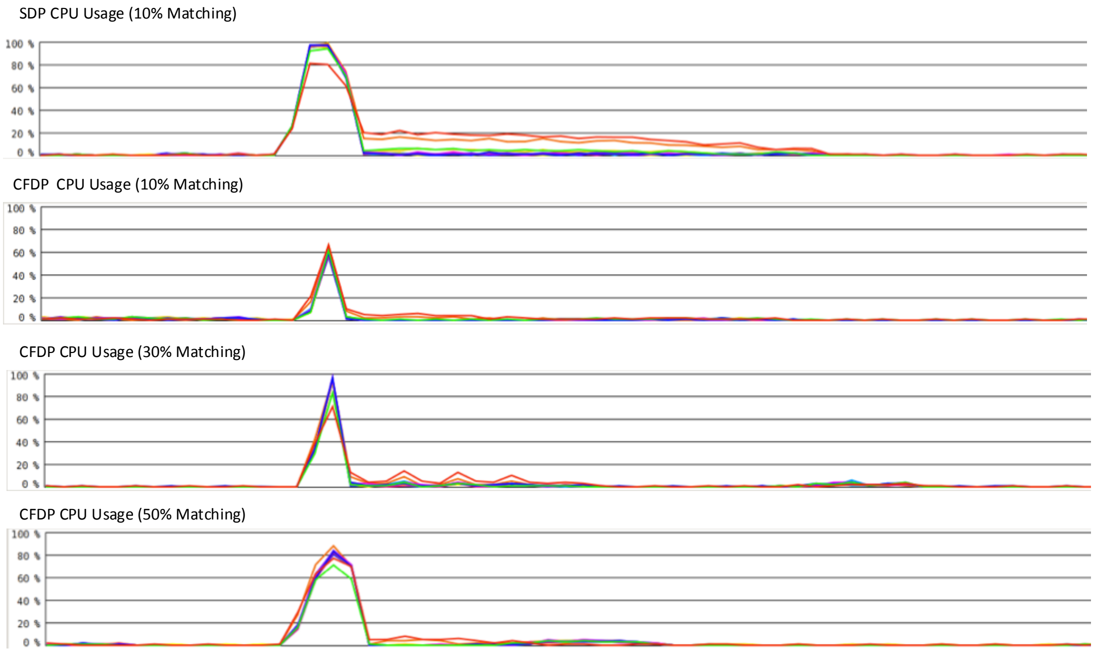


Figure 12: CFDP and SDP CPU Usage Comparison

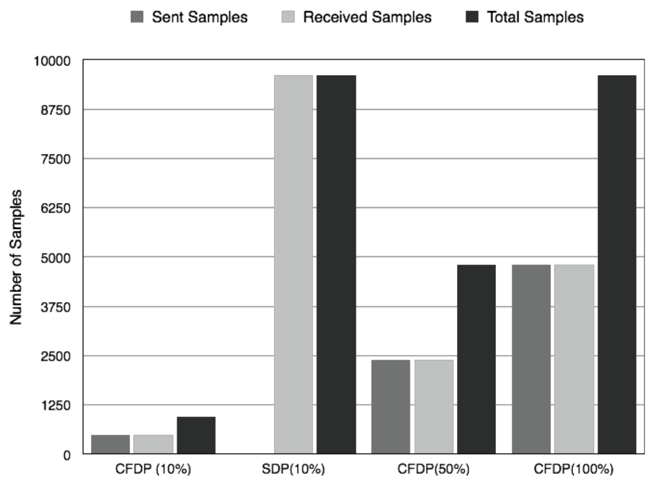


Figure 13: Number of Sent/Received Discovery Messages

using SDP is still larger than any case of CFDP because the number of received samples is always large.

In this test the number of received transfers for SDP is 9,600, which is 10 times larger than CFDP. Even for 100% matching ratio of CFDP, CFDP uses less messages than SDP because filtering expressions for publication and subscription exist separately, and discovery messages for the same kind of endpoints are automatically filtered as they do not need to communicate.

The size of *publication DATA* is 292 Bytes and the size of *subscription DATA* is 260 Bytes. Based on these sizes, we can calculate memory and network usage approximately. The total transferred bytes of CFDP (10%) is 264.96 KB and the total transferred bytes of SDP is 2808.4 KB. The estimated memory consumed per participant for CFDP is thus 124.8 KB, while SDP requires 2,496 KB.

4.4 Discussion

The following is a summary of the lessons we learned from this research and the empirical evaluations.

- **CFDP is more efficient and scalable than SDP**. The experimental results show that CFDP is more efficient and scalable than SDP in terms of discovery completion time, as well as in the use of computing and network resources. In particular, for discovery completion time, CFDP is 5 times faster than SDP on average and minimum discovery completion time is 15 times faster when the matching ratio is 0.1 (10%). For CFDP the computing and network usage linearly decreases as the matching ratio decreases. CFDP therefore disseminates and processes discovery of peers and endpoints more efficiently and scalably than SDP.

- **CFDP's current lack of support for multicast can impede scalability**. Our empirical tests indicated that the number of messages sent by SDP is smaller than those sent by CFDP since CFDP does not use multicast to filter messages on the data publisher side. SDP can seamlessly use multicast because only a single multicast address is needed for all participants to use. For CFDP, however, each content filter used by CFDP will need a separate multicast address. To overcome this limitation, our future work will enhance CFDP to support multicast thereby reducing the number of discovery messages sent by delegating the overhead to network switches. This approach will group peers with a set of multicast addresses by topic names so that built-in discovery DataWriters will publish data only to assigned multicast channels (groups). We also plan to leverage *Multi-channel DataWriters*, which is a DataWriter that is configured to send data over multiple multicast addresses according to the filtering criteria applied to the data [14]. By using this feature, the underlying DDS middleware evaluates a set of filters configured for the DataWriter to decide which multicast addresses to use when sending the data.

- **Instance-based filtering can help to make CFDP scalable in a large-scale system with a small set of topics**. DDS supports a *key* field in a data type that represents a unique identifier for data streams defined in a topic. A data stream identified by a key is called *instance*. The current CFDP filters discovery messages based on topic names, which limits its scalability in a system where most data streams are differentiated by a key of a topic, rather than by a topic itself. For example, a regional air traffic management system may have many endpoints that exchange data by using a single topic (such as flight status), but are interested in only a specific set of flights that are identified by their keys. In such a system, even though all endpoints are involved in the same topic, they do not need to be discovered by each other because its interest is not based on the topic name, but on the key value of the topic. In future work we will enhance CFDP to filter discovery messages based on topic names as well as instance IDs (keys). This enhancement should provide performance benefits for DRE systems that contain numerous endpoints and instances with a single or less number of topics.

5. RELATED WORK

This section compares and contrasts our CFDP DDS peer discovery mechanism with other discovery mechanisms reported in the literature. Several DDS discovery protocols have been developed to satisfy different system requirements and deployment scenarios. The *Simple Discovery Protocol* (SDP) [10] is the standard DDS discovery protocol. SDP is a decentralized and distributed approach to discover remote peers and their endpoints as each peer independently manages discovery information. It requires no effort in discovery configurations and avoids a single point of failure. The motivation for our work on the CFDP approach is to overcome the limitations of scalability inherent in the standard SDP approach.

OMG DDS also supports a centralized approach [14] [8], which requires a dedicated service to manage all of the participants and endpoints in a domain. This approach can be more scalable than SDP in certain configurations because every peer in a domain need not exchange discovery messages with all other peers in a domain, but only communicate with the central node where the dedicated service runs. This centralized scheme, however, has several drawbacks. First, its centralized design can become a single point of failure. Second, if the dedicated service is overloaded, the performance of the discovery process of peers in a system can deteriorate considerably.

To avoid the run-time overhead incurred by both the decentralized and centralized discovery protocols, an alternative is a static discovery protocol [13]. In this model users manually configure the discovery information of peers and their endpoints at design- and deployment-time, which requires significant configuration efforts. Such approaches can be useful, however, for closed DRE systems that are deployed in networks with limited resources because no additional resources are used during the run-time discovery process. The key distinction of this static approach with our work on CFDP is that the latter is not a static approach and operates in an open environment.

An improvement to SDP is presented in [16] by utilizing bloom filters to address scalability problems incurred in large-scale DRE systems. The peers send bloom filters to other peers, where the bloom filter is a summary of endpoints deployed in a peer. Peers use the received bloom filters to check if the matching endpoints are in the set represented by the filter. The peers store information about all endpoints, but leverage the smaller size enabled by bloom filters to use network and memory resources more efficiently. Although this related work has similar goals as CFDP (*i.e.*, both approaches attempt to solve the same problem), each approach is designed and implemented differently. In particular, rather than using a bloom filter, CFDP uses a content filter topic (CFT) to filter unmatched endpoint discovery messages.

In [18], the authors investigate the challenges of using DDS in large-scale, network-centric operations and suggest an adaptive discovery service framework to meet key performance requirements. This work also mentions the scalability problem of DDS discovery, but focus on the discovery scalability problems incurred in WANs. Likewise, the work presented by [5] outlines an extension to the IETF *REsource Location And Discovery* (RELOAD) protocol [4] for content delivery in WAN-based IoT systems using DDS. The authors conducted experiments with 500 to 10,000 peers over a simulated network to show its scalability. Although this paper addressed the discovery scalability issue of DDS, their approach centers on a structured P2P overlay architecture in WANs, which is different from our work on CFDP, which is based on an unstructured P2P scheme.

Discovery is an important issue for the domain of peer-to-peer (P2P) systems, which can be classified into structured P2P and unstructured P2P schemes [6]. The structured P2P scheme, such as Chord [17] and Pastry [12], assigns keys to data and organizes peers into a graph (a distributed hash table) that maps each data key to a peer, and therefore realizes efficient discovery of data using the keys. The standard discovery approach for DDS in LANs based on SDP can be classified as an unstructured P2P scheme since it organizes peers in a random graph by allowing anonymously joining and leaving participants via multicast. The unstructured P2P scheme is not efficient compared to the structured one because discovery messages must be sent to a large number of peers in the network to build a graph of peers that remain anonymous to each other, however, this approach is needed to support the spatio-temporal decoupling of peers. Since our CFDP solution is designed to improve SDP, it operates in the unstructured P2P environment.

A taxonomy that compares and analyzes existing technologies for discovery services in *Ultra-large-Scale* (ULS) systems [7] is presented in [3]. The authors evaluate discovery services along four dimensions: *heterogeneity*, *discovery QoS*, *service negotiation*, and *network scope and type*. To support interoperability between heterogeneous systems, the OMG DDS supports diverse operating platforms (*i.e.* Linux(x86), Windows, and VxWorks), and also has standardized the `Real-TimePublish-Subscribe` (RTPS) protocol [10] for different DDS implementations by vendors. DDS uses built-in DDS entities, such as topics and endpoints, to discover peers, as explained in Section 2.2. DDS' QoS policies can in turn be used by these entities to support QoS of discovery. For service negotiation, DDS discovery protocols compare the requested and offered QoS policies by data producers and data consumers at the endpoint discovery phase. DDS discovery mechanisms originally focused on the *Local Area Network* (LAN) scope. Recent research [2] has broadened their scope to support *Wide Area Networks* (WANs) by deploying additional capabilities, such as the DDS Routing Service [15] that transform and filter local DDS traffic to different data spaces (*i.e.*, network domain).

6. CONCLUDING REMARKS

This paper motivated the need to improve the efficiency and scalability of the standard *Simple Discovery Protocol* (SDP) [10] used for DDS applications. We then presented the design and implementation of our *Content-based Filtering Discovery Protocol* (CFDP), which enhances SDP for a large-scale systems by providing a content filtering mechanism based on standard DDS features to eliminate unnecessary discovery messages for participants according to matching topic names and endpoint types. We also analyzed the results of empirical tests to compare the performance of SDP and CFDP, which indicate that CFDP is more efficient and scalable than SDP in terms of CPU, memory, and network usage. Our future work will address the limitations with CFDP presented in Section 4.4.

7. ACKNOWLEDGMENTS

This work is supported in part by the DoE Phase I SBIR grant award DE-SC0009641 and NSF CAREER CNS 0845789. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DoE and NSF.

8. REFERENCES

[1] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[2] A. Hakiri, P. Berthou, A. Gokhale, D. Schmidt, and T. Gayraud. Supporting End-to-end Scalability and Real-time Event Dissemination in the OMG Data Distribution Service over Wide Area Networks. *Elsevier Journal of Systems Software (JSS)*, 86(10):2574–2593, Oct. 2013.

[3] J. Hoffert, S. Jiang, and D. C. Schmidt. A Taxonomy of Discovery Services and Gap Analysis for Ultra-large Scale Systems. In *Proceedings of the 45th annual southeast regional conference*, pages 355–361. ACM, 2007.

[4] C. Jennings, S. Baset, H. Schulzrinne, B. Lowekamp, and E. Rescorla. Resource Location and Discovery (RELOAD) Base Protocol. *REsource*, 2013.

[5] J. M. Lopez-Vega, G. Camarillo, J. Povedano-Molina, and J. M. Lopez-Soler. RELOAD extension for data discovery and transfer in data-centric publish–subscribe environments. *Computer Standards & Interfaces*, 36(1):110–121, 2013.

[6] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A Survey and Comparison of Peer-to-peer Overlay Network Schemes. *IEEE Communications Surveys and Tutorials*, 7(1-4):72–93, 2005.

[7] L. Northrop, P. Feiler, R. Gabriel, J. Goodenough, R. Linger, R. Kazman, D. Schmidt, K. Sullivan, and K. Wallnau. Ultra-Large-Scale Systems: Software Challenge of the Future. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, June 2006.

[8] OCI. Open DDS Developer's Guide. `http://download.ociweb.com/OpenDDS/OpenDDS-latest.pdf`, 2013.

[9] OMG. The Data Distribution Service Specification, v1.2. `http://www.omg.org/spec/DDS/1.2`, 2007.

[10] OMG. The Real-Time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification, V2.1. `http://www.omg.org/spec/DDS-RTPS/2.1`, 2010.

[11] G. Pardo-Castellote. OMG Data-Distribution Service: Architectural Overview. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 200–206. IEEE, 2003.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-scale Peer-to-peer Systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[13] RTI. Limited-Bandwidth Plug-ins for DDS. `http://www.rti.com/docs/DDS_Over_Low_Bandwidth.pdf`, 2011.

[14] RTI. RTI Connext DDS User's Manual. `http://community.rti.com/rti-doc/510/ndds.5.1.0/doc/pdf/RTI_CoreLibrariesAndUtilities_UsersManual.pdf`, 2013.

[15] RTI. RTI Routing Service User's Manual. `http://community.rti.com/rti-doc/510/RTI_Routing_Service_5.1.0/doc/pdf/RTI_Routing_Service_UsersManual.pdf`, 2013.

[16] J. Sanchez-Monedero, J. Povedano-Molina, J. M. Lopez-Vega, and J. M. Lopez-Soler. Bloom Filter-based Discovery Protocol for DDS Middleware. *Journal of Parallel and Distributed Computing*, 71(10):1305–1317, 2011.

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *ACM SIGCOMM Computer Communication Review*, volume 31, pages 149–160. ACM, 2001.

[18] N. Wang, D. C. Schmidt, H. van't Hag, and A. Corsaro. Toward an Adaptive Data Distribution Service for Dynamic Large-scale Network-centric Operation and Warfare (NCOW) Systems. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7. IEEE, 2008.