# Establishing Secure Interactions Across Distributed Applications in Satellite Clusters

Subhav Pradhan, William Emfinger, Abhishek Dubey, William R. Otte, Daniel Balasubramanian, Aniruddha Gokhale and Gabor Karsai*

*ISIS/EECS, Vanderbilt University
Nashville, TN 37235, USA
{pradhasm,emfinger,dabhishe,wotte, daniel,gokhale,gabor}@isis.vanderbilt.edu

Alessandro Coglio†

†Kestrel Institute,
Palo Alto, CA 94304, USA
coglio@kestrel.edu

*Abstract*—**Recent developments in small satellites have led to an increasing interest in building satellite clusters as open systems that provide a "cluster-as-a-service" in space. Since applications with different security classification levels must be supported in these open systems, the system must provide strict information partitioning such that only applications with matching security classifications interact with each other. The anonymous publish/subscribe communication pattern is a powerful interaction abstraction that has enjoyed great success in previous space software architectures, such as NASA's Core Flight Executive. However, the difficulty is that existing solutions that support anonymous publish/subscribe communication, such as the OMG Data Distribution Service (DDS), do not support information partitioning based on security classifications, which is a key requirement for some systems. This paper makes two contributions to address these limitations. First, we present a transport mechanism called *Secure Transport* that uses a lattice of labels to represent security classifications and enforces Multi-Level Security (MLS) policies to ensure strict information partitioning. Second, we present a novel discovery service that allows us to use an existing DDS implementation with our custom transport mechanism to realize a publish/subscribe middleware with information partitioning based on security classifications of applications. We also include an evaluation of our solution in the context of a use case scenario.**

## I. Introduction

A cluster of satellites, whose overall mission functionality is distributed across individual modules, provides a resilient and highly dynamic architecture [?]. Such an architecture of satellite clusters allows applications to take advantage of the heterogeneity and redundancy of computing and sensing devices available on different modules in the cluster. The heterogeneity of modules allows the system to provide a wide variety of functionality while the redundancy becomes useful during failures and outages of modules. Several existing and future missions use this type of architecture including NASA's Edison Demonstration of SmallSat Networks, TANDEM-X, PROBA-3, and PRISMA from Europe. In each of these missions, the cooperating fractionated satellites are expected to provide the foundations for truly distributed software applications.

In essence, the satellite clusters described above can be thought of as "open" systems that provide a "cluster-as-a-service" in space. Applications can be supplied by different organizations, each of which may have different security requirements and classifications. These applications should not be allowed to interact arbitrarily, but rather only if they have compatible security classifications. For example, an application with classified information should not be allowed to send it to an unauthorized application. Since the applications themselves may not be fully trusted, the platform itself must provide a secure execution environment that prevents arbitrary flows of information between applications.

Applications in such distributed systems can interact with each other using a variety of communication patterns, such as the asynchronous method invocation and anonymous publish/subscribe patterns. For applications like flight software, which need to distribute data without explicitly knowing which applications wish to receive it, the publish/subscribe interaction pattern [?] can be a solution. In this pattern, a sender publishes messages without any knowledge of which applications may be subscribing to them. Similarly, a receiver subscribes to messages without any knowledge of which applications are publishing the messages.

One notable example for the use of the publish/subscribe pattern in spacecraft software is NASA's Core Flight Executive (CFE), which is a portable platform-independent framework used as the basis for flight software for satellite data systems. The CFE uses publish/subscribe to facilitate communication between multiple applications. A review of the CFE revealed that the publish/subscribe style architecture not only allowed distributed development and easy integration of applications but also allowed applications to be encapsulated, which improved abstraction, flexibility, reuse and division of concerns [?]. Thus, supporting this pattern in space systems is highly desirable.

Unfortunately, existing technologies that support publish/subscribe communication, such as the OMG Data Distribution Service (DDS) [?], do not have a comprehensive security model capable of partitioning information flows based on the security classifications of applications, which is a key requirement for open and distributed satellite clusters platforms. The distributed system developers for satellite clusters are now faced with a dilemma: there is a compelling argument to use publish/subscribe in the clusters, however, existing standardized publish/subscribe technologies do not support the partitioning of information flows based on a strict security model. The primary research contribution of this paper is to address this key limitation in a standardized publish/subscribe middleware in a way that does not need any non-standard and invasive changes to the middleware so that the enhanced but standards-compliant solution can continue to be used in a variety of domains.

The research presented in this paper makes two primary contributions. Our first contribution is motivated by the need to remain standards-compliant and hence comprises a novel mechanism called *Secure Transport (ST)*, which is a network transport layer that enforces information flow partitions based on security classifications. ST uses Multi-Level Security (MLS) labels [**?**] to represent security classification; and therefore, MLS policies are used to enforce information partitioning based on a set of linearly ordered hierarchical classification levels and non-hierarchical need-to-know categories. The standard MLS policy states that information can only flow on the same level or from a lower to higher level according to the dominance relation. In addition to supporting information partitioning based on this MLS policy, ST also restricts any communication topology that can be established between any two application processes without any governance from another process with elevated system privileges.

Our second contribution is motivated by the need for the standards-compliant middleware to leverage the new mechanisms, and hence comprises a novel discovery service that creates ST information flows between matching publisher and subscriber endpoints of different applications. This is required because without ST information flows, the publishers and subscribers cannot communicate with each other. To realize this service, we extended the centralized discovery mechanism supported by OpenDDS [**?**] such that it becomes a privileged system entity responsible for authorizing and establishing ST information flows between application components with security labels. This change does not interfere with the OMG DDS standard. Furthermore, to make sure that this discovery service is not abused, we ensure that only privileged entities of the system can interact with it.

**Impact of our work on space systems**: Distributed space systems, such as fractionated satellites, provide tremendous flexibility and economic benefits since smaller and lighter satellites are cheaper to build, launch, and maintain. It also results in a system, which hosts distributed applications that interact with each other via powerful and reusable interaction paradigms including the anonymous publish/subscribe pattern.

Regardless of the communication patterns used by applications, security is of critical importance since (a) space systems are extremely expensive and therefore cannot afford security compromises with long term effects on the system, (b) shared space systems are used by companies that are extremely sensitive about their data and therefore require a strict security model. Our research demonstrates how a standardized publish/subscribe middleware can be extended without invasive changes to the standard and applied in the context of open, distributed satellite clusters to support the strict security model of partitioned information flows between publishers and subscribers.

The rest of this paper is organized as follows. Section II provides a brief description of our system model, presents an overview of the current OMG DDS standard, and finally describes the problem and challenges that we solve in this paper. Section III provides a description of Secure Transport and how it supports information partitioning through the use of MLS labels. Section IV describes, in detail, our solution architecture for a novel, secure, discovery mechanism. Section V presents experimental results, and Section VI presents related research along with a comparison to our work. We conclude in Section VII.

## II. Problem Description

This section describes the problem we address in this research including the system model we assume and an overview of technologies used to realize the solution.

### A. System Model

We consider a cluster of satellites where each satellite has a computing node that runs a copy of the **D**istributed **RE**altime **M**anaged **S**ystem (DREMS) platform [**?**], [**?**]. DREMS is a comprehensive information architecture that comprises an operating system, middleware, and a component framework. Applications are composed from reusable components, which are hosted inside 'actors'. Actors are the building blocks of this information system and are similar to processes in traditional operating systems except that the Actor identifiers are unique across nodes and are not lost even after the death of the Actor. Actors can be of two types: (a) Application Actors, and (b) Platform Actors. Application Actors make up the mission-specific application functionality and can be dynamically installed or removed. Platform Actors are extensions of the OS in the sense that they serve to provide long running services that are secure and therefore can invoke privileged system calls. Platform Actors are part of the Trusted Computing Base (TCB) [**?**]. Applications hosted in our platform cannot bypass the TCB and access resources, such as the network.

To provide a higher level of abstraction to develop reusable and composable application functionality, a middleware layer provides an abstraction of lower-level operating system services, simplifying their use and specifying well-defined patterns for both local and remote communication. Primarily, both point-to-point and anonymous publish/subscribe interactions are available. The publish/subscribe interactions are provided using a modified version of OpenDDS [**?**], which is an open source implementation of the OMG DDS specification.

### B. OMG DDS Overview

The OMG Data Distribution Service (DDS) specification [**?**] defines a data-centric communication standard for communication between DDS entities (information producers and consumers) in a wide variety of environments. A DDS application consists of *publishers* and *subscribers,* where publishers use *data writers* to produce data while subscribers use *data readers* to consume data. The DDS specification also supports many Quality of Service (QoS) policies which allows application

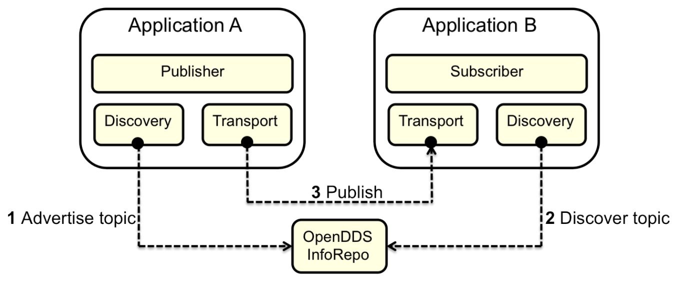


Fig. 1: OpenDDS Centralized Discovery Mechanism

developers to fine tune non functional properties of a DDS middleware, and hence the communication between publishers and subscribers.

From an application's perspective, publishers and subscribers are anonymous; yet they need to communicate with each other, which needs a mechanism for them to discover each other. To discover and match publishers and subscribers (and therefore data writers and data readers), the DDS middleware uses a *discovery service*. Even though the exact implementation of this discovery service varies from one implementation to another, a common requirement for this service is to be able to discover matching publishers and subscribers and to establish connections between them. To perform this matching, the discovery service uses *topics*. A topic is a data type that serves the fundamental mechanism to match up publishers and subscribers. Communication between a data writer and a data reader does not occur unless the topic published by the writer matches the topic subscribed to by the reader. Figure 1 illustrates the centralized discovery mechanism used by OpenDDS.

## C. Problem Statement

The main problem we address in this paper is the need to support secure interactions between distributed applications that use the publish/subscribe pattern in open distributed systems. We have identified the following two primary challenges that we resolve in this paper:

*1) **Challenge 1: A mechanism for secure interaction**:* To support secure interactions between applications with varying security classifications (and hence security labels), we need a mechanism that permits interactions between applications with matching security labels only. This means that an application with a higher security label should be able to receive information from another application that has the same or a lower security label, but an application with a lower security label must not be able to receive information from an application with a higher security label. Primarily, our focus is on supporting such a capability that can be leveraged by OMG DDS without making invasive changes to the standard. Section III describes our solution involving a secure transport mechanism. We do not consider *non interference* in this work, which is a stronger policy and often requires performance isolation.

*2) **Challenge 2: Secure publish/subscribe interaction**:* Since we use DDS middleware to provide publish/subscribe

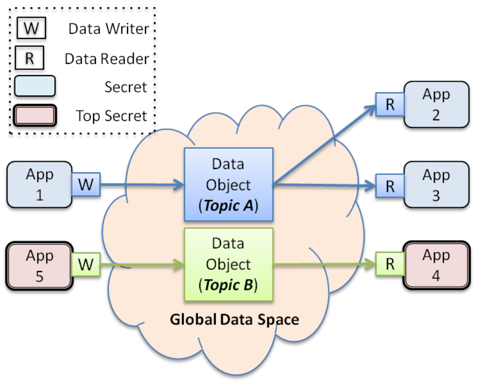


Fig. 2: Topic based application interaction in OMG DDS

capabilities to applications with different security classifications, we require these interactions be secure and provide appropriate information partitioning. However, the OMG DDS specification, as it stands currently, does not support any specific security policy, and therefore existing DDS implementations lack a well-defined security model[1]

Existing DDS entities, such as topics and partitions cannot be used as mechanisms to enforce information partitioning based on security labels since applications associated with the same topic can have different security classifications. A naive solution would be to create a topic for each security label, as shown in Figure 2. However, this approach has a number of problems. First, it makes the applications responsible to subscribe to topics that are appropriate to their security level. Second, this approach also relies on higher security level applications to correctly refrain from publishing samples to topics to which lower security level applications are subscribed to. Both these shortcomings are unworkable as we cannot trust applications to behave correctly.

Therefore, to solve this problem, we need to leverage the solution for *Challenge 1*, described in Section II-C1. Specifically, we require a mechanism that is capable of using the solution for *Challenge 1* in order to establish secure publish/subscribe communications between applications with correct security labels. Section IV describes our solution.

## III. Resolving Challenge 1: Secure Transport

This section presents Secure Transport (ST), which is a novel transport mechanism implemented in the OS kernel that (a) restricts the communication pathways that can exist between any two processes on the same computing node or on different computing nodes, (b) supports both unicast as well as multicast transfers, (c) supports message authentication before transmission and reception according to rules of a Multi-Level Security (MLS) policy [**?**], [**?**], [**?**], and (d) provides packet level encryption for messages using IPSec [2].

[1]There is a DDS security specification in development [**?**], however it relies on the correct operation of the application and middleware to configure and enforce these policies, which are not trusted in our model. This is discussed in more detail in Section VI.

[2]The IPSec implementation in ST is the subject of ongoing work

MLS defines a policy based on partially ordered security labels that assign classifications and need-to-know categories to all information that may pass across process boundaries. This policy ensures that information is allowed to flow from a producer to a consumer in DDS if and only if the label of the consumer is greater than or equal to that of the producer. Both the allowed communication topology and the MLS policy labels must be configured for each task by the secure discovery mechanism, which uses processes with elevated system privileges to do so. By designing a solution that resides at the level of a transport mechanism, we shield higher level middleware, such as OMG DDS, from invasive changes while at the same time providing them an opportunity to leverage these new mechanisms.

Secure Transport (ST) comprises the following key mechanisms described below.

### A. Endpoints

Endpoints are the basic communication resources used by applications to transmit and receive messages; they are analogous to socket handles in the traditional BSD socket APIs. Like traditional sockets, user space programs pass an endpoint identifier to the `send` and `receive` system calls. Unlike traditional sockets, however, unprivileged tasks may not arbitrarily construct endpoints that allow for inter-process communication with other tasks; such endpoints must be explicitly configured by a privileged Actor which is part of a trusted system configuration infrastructure. All endpoints are configured with a set of security labels that can be used for sending messages through that endpoint.

Endpoints are separated into four different categories with different restrictions on their creation and use; for the purposes of this discussion, we will describe only the two endpoint classes that are used for Inter-Process Communication (IPC):

- **Local Message Endpoints (LME)**: Local message endpoints are the basic method for IPC and may be used to send messages to other Actors hosted by the same operating system instance. These endpoints must be configured by the trusted system configuration infrastructure and are subject to restrictions placed by flows and security rules.
- **Remote Message Endpoints (RME)**: Similar to LMEs, RMEs are a mechanism for network IPC between Actors, but may be used to communicate with Actors hosted by different operating system instances.

### B. Flows and Message Transfer Rules

Communication in ST is allowed between two LMEs or RMEs if and only if there exist mutually compatible *flows* on each endpoint. A flow can be thought of as a logical pipe between two endpoints and determines the direction in which messages can travel. It provides system integrators the ability to specify the actors that are allowed to share messages. The actual transfer of the message is further restricted by the MLS rules (see below).

More concretely, a flow that is assigned to an endpoint is a connectionless association with an endpoint owned by a designated Actor. This association determines if the local endpoint is allowed to send or receive messages with the remote endpoint. Unicast flows connect a source endpoint to a destination endpoint; multicast flows connect a source endpoint to multiple destination endpoints.

In all cases, the flow assignment between two endpoints must be mutual in order for communication to succeed. Additionally, each message must be marked with the specific label that indicates the security classification of that message. Message transmission is allowed if and only if the following rules are satisfied. We refer the readers to [**?**] for a full list of formal MLS rules.

- The label of the message must be within the label set of the sender endpoint.
- The sender must have an outbound flow to the recipient, and the recipient must have a inbound flow from the sender.
- The receiver's endpoints label must be either at the same classification level or at a higher classification level of the received message. Thus, a lower classification application cannot extract information from a higher classification application on the reply path for a two-way communication because the labels on the return path do not satisfy the MLS rules.

Performing message exchange via endpoints and flows enables decoupling between senders and receivers, which operate only on their local endpoints without explicit knowledge of the flows attached to those endpoints. For example, the flow connecting a client to a failed server can be switched over to an alternative server transparently to the client.

## IV. Resolving Challenge 2: A Secure Discovery Mechanism

Recall from Challenge 2 that the DDS publish/subscribe communication was required to be secure and support the information partitioning. To shield the DDS standard from invasive changes, our security mechanism was designed at the transport layer, in ST. This section therefore describes how our DDS infrastructure leverages the ST mechanism. To that end we first need to understand how the Deployment and Configuration (D&C) infrastructure in DREMS works because it is responsible for assigning the security labels, setting up endpoints and the flows between communicating actors and ultimately playing a key role in the secure discovery. Subsequently, we present a detailed description of our novel discovery mechanism that establishes secure publish/subscribe interactions between applications using the ST.

### A. Overview of the D&C Infrastructure

The D&C infrastructure is responsible for the deployment and configuration of component-based applications. Once an application is deployed, the D&C infrastructure is also responsible for managing the application's lifecycle throughout its lifetime. The D&C infrastructure is an implementation of the OMG D&C specification [**?**] and it is composed of multiple *Deployment Managers (DMs)* at different levels of the cluster.

TABLE I: Entities involved in the Secure Discovery Mechanism

| Entity | Functionality |
|---|---|
| DDS Entity | This represents the various DDS related entities such as domain participants, data writers and data readers. These entities are part of a user's component-based applications and therefore hosted inside their respective Component Servers (CSs). |
| DCPSInfo | A singleton used to keep track of various DDS entities created by different applications and inform the DDS middleware. |
| CS ORB | This is the default CORBA Object Request Broker (ORB) [?] that is created when a CS is instantiated by a Node Deployment Manager (NDM). This ORB is used by the CS to communicate with its parent NDM. |
| Discovery Callback | All new data readers and data writers register a callback object, which is used by the discovery mechanism to provide information regarding data reader/data writer matches. |
| DictM Proxy | Since NDMs do not have access to the Dictionary Manager (DictM), this proxy is used for communicating with the DictM, which is collocated with the CDM. |
| Callback Proxy | The Callback proxy is created by the DictM proxy. Upon creation of the Callback proxy, the DictM proxy provides this information to the DictM. This is important because the DictM provides the data reader/data writers match information to the Callback proxy, which in turn uses this information to establish the ST flows between endpoints used by matching data readers and writers. The Callback proxy uses the ST interfaces to create these flows. Note that the ST flows must be established before the DDS middleware can establish connection between data readers and data writers by using their respective callback objects. |
| DM ORB | This is the ORB used by the NDM and CDM to communicate with each other. |
| DictM | The Dictionary Manager (DictM) is a Platform Actor collocated with the CDM. It acts as a central information repository, which is notified every time a new data reader or data writer is created. Upon creation of a new data writer, the DictM updates information content and upon creation of a new data reader, the DictM finds a matching writer (if any) and propagates this information to the callback objects of the created data readers. |

In a satellite cluster there is exactly one Cluster Deployment Manager (CDM), which is responsible for orchestrating the deployment and configuration (D&C) of applications across multiple nodes in the cluster. Each node has a single Node Deployment Manager (NDM) that is responsible for performing node-specific D&C activities. The CDM and NDM are Platform Actors and are therefore considered to be part of the TCB. Finally, each node can have multiple Component Servers (CS), which are spawned by NDMs as per requirement. All parts of an application, *i.e.*, component servers, components, and secure transport endpoints are created by the D&C infrastructure according to a deployment plan, which is created by a trusted system integrator. The trusted system integrator is also responsible for assigning MLS labels to different applications. A more detailed description of this D&C infrastructure can be found in our earlier work [?].

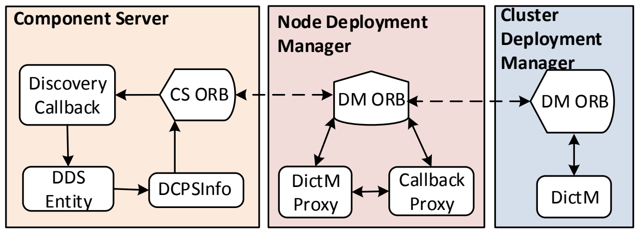


Fig. 3: Discovery Architecture. Table I describes the entities shown in this figure.

## B. An Architecture for Secure Discovery Mechanism

The discovery mechanism discussed in this section leverages existing discovery capabilities provided by OpenDDS. OpenDDS supports two forms of discovery mechanisms: (1) using a centralized information repository called the *InfoRepo*, and (2) using a peer-to-peer approach where participating DDS applications discover each other in a collaborative way rather than using a centralized information repository.

The peer-to-peer discovery approach requires all applications in a network be able to communicate each others existence in order to perform discovery. This is not a workable solution for our purpose as the communication restrictions enforced by ST mechanism preclude any two unprivileged application processes spontaneously connecting with each other. In addition, an application with lower security label must not be able to discover the existence of other applications with a higher security label. Since we cannot use the peer-to-peer discovery mechanism, our solution extends the centralized discovery service provided by OpenDDS. Our extended capabilities require the following functionalities:

- The discovery mechanism should be able to establish Secure Transport information flows between data writers and data readers of applications with matching security labels. Since ST is a transport mechanism implemented in the kernel itself, it requires the discovery mechanism to have elevated system privileges to create ST endpoints and establish ST information flows.
- In addition to the discovery mechanism itself having system privileges, it should only interact with other processes with system privileges. This prevents the centralized repository from being a target of "data-at-rest" attacks.
- Normally, a discovery mechanism only checks for topic and relevant Quality of Service (QoS) parameters when matching data writers and data readers. However, since we are using the concept of security classification represented by security labels, we require that the discovery mechanism also check these security labels during the discovery process.

To address these challenges and since we build on top of OpenDDS' centralized approach, a centralized discovery mechanism called the *Dictionary Manager (DictM)* is used. The DictM is a Platform Actor and therefore has system privileges to establish Secure Transport information flows and it only interacts with other Platform Actors.

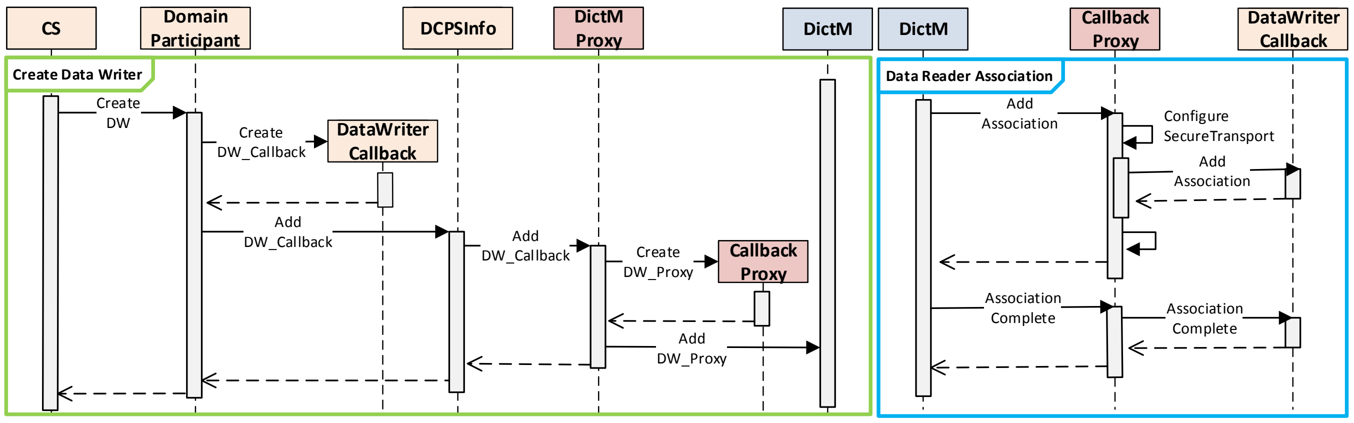


Fig. 4: Sequence Diagram illustrating the process of (a) Data Writer creation, and (b) Data Reader association

Figure 3 presents the architecture for the secure discovery mechanism showing how it leverages the D&C infrastructure (and hence the TCB). Table I enlists the functionalities of these entities. Figure 4 presents sequence diagrams illustrating (1) creation of a data writer, and (2) association of a data reader with an existing data writer.

As shown in Figure 4, to create a data writer, first the Component Server creates a DDS domain participant which in turn is used to create the data writer. Both the domain participant and data writer are DDS Entities. The domain participant also creates a data writer callback, which is the Discovery Callback that will later be used to inform the data writer about matching data readers. Once the data writer callback object is created, this information is propagated to the DCPSInfo and DictM Proxy, which in turn creates a Callback Proxy and sends this information to be stored in the DictM.

Data reader creation follows the same pattern as that of a data writer. Once the data reader is created and this information is propagated to the DictM, it finds the matching data writer and adds the association. This process is also shown in Figure 4. Adding an association consists of configuring a ST flow between endpoints of the matched data reader and data writer. Once the ST flow is established, the association is complete and now the DDS middleware can establish a connection between the data writer and the data reader.

## V. Evaluation

This section evaluates our solution in the context of a use case scenario. Since the Secure Transport (ST) leverages IPv6, we also compare the effective performance of ST against native IPv6; both transmitted over the same physical medium. This evaluation is important since ST will be used for all the secure communication by our system and hence the cost of using such a mechanism must be understood prior to its use.

### A. Experimental Setup

Figure 5 presents the setup of our experiment comprising two applications. These applications use DDS pub/sub for communication. Each application consists of a *publisher* and *subscriber*. App-1 is hosted on node-1 and has security label *unclassified*, whereas, App-2 is hosted on node-2 and has the security label *secret*. App-2's security label thus dominates the label of App-1, and hence only App-1 is allowed to send information to App-2.

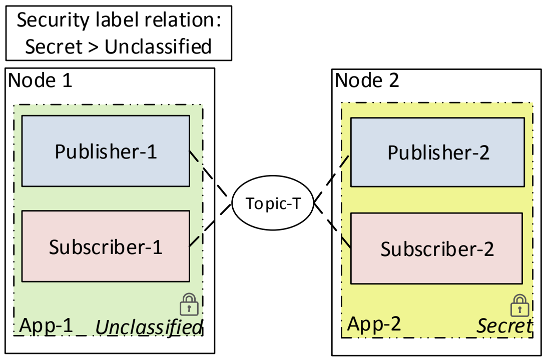


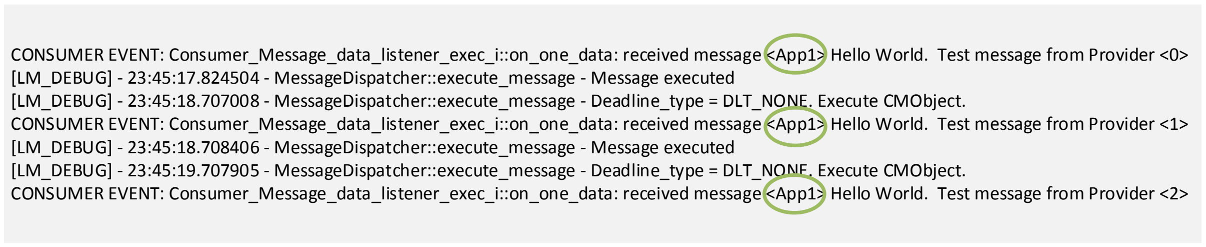
Fig. 5: Use case scenario: Two DDS applications with different security labels

Based on the setup mentioned above, the subscriber in App-2 should be able to receive messages published by publishers in both App-2 and App-1. However, the subscriber in App-1 should only be able to receive messages published by a publisher in App-1 since App-2 is publishing messages with higher security label. To demonstrate this behavior, Figures 6(a) and 6(b) present log message snippets[3] captured during the execution of the above-described use case with the appropriate App shown circled.

### B. Secure Transport Network Utilization

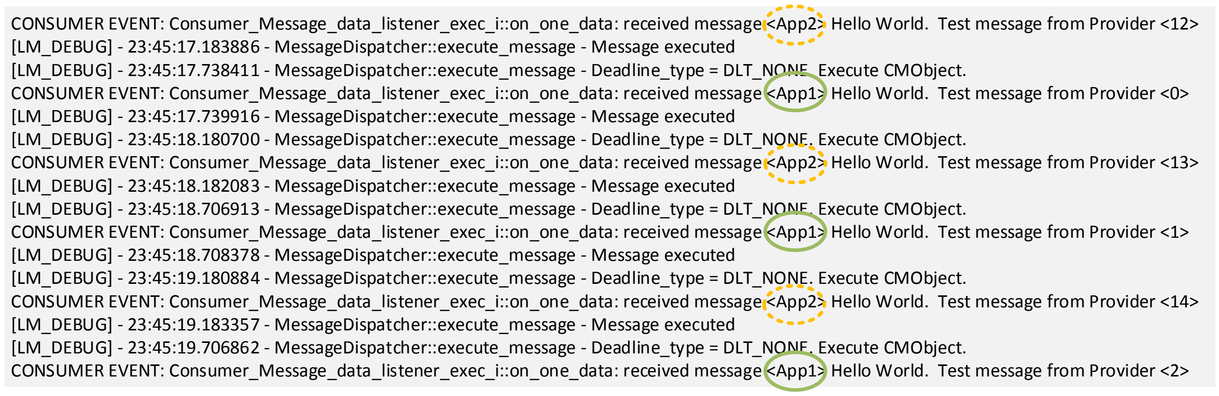
The Secure Transport (ST) mechanism presented previously in Section III is built on top of IPv6. With any communication protocol, performance is a key concern, so the effective performance (i.e., network utilization) of ST versus native IPv6 transmitted over Ethernet was evaluated. First the overhead of

[3]These log message snippets have been slightly modified in order to exclude irrelevant log messages.



```
CONSUMER EVENT: Consumer_Message_data_listener_exec_i::on_one_data: received message <App1> Hello World.  Test message from Provider <0>
[LM_DEBUG] - 23:45:17.824504 - MessageDispatcher::execute_message - Message executed
[LM_DEBUG] - 23:45:18.707008 - MessageDispatcher::execute_message - Deadline_type = DLT_NONE. Execute CMObject.
CONSUMER EVENT: Consumer_Message_data_listener_exec_i::on_one_data: received message <App1> Hello World.  Test message from Provider <1>
[LM_DEBUG] - 23:45:18.708406 - MessageDispatcher::execute_message - Message executed
[LM_DEBUG] - 23:45:19.707905 - MessageDispatcher::execute_message - Deadline_type = DLT_NONE. Execute CMObject.
CONSUMER EVENT: Consumer_Message_data_listener_exec_i::on_one_data: received message <App1> Hello World.  Test message from Provider <2>
```

(a) App-1 log message snippet which shows that App-1 only receives messages published by itself and not from App-2 since the latter has higher security label



```
CONSUMER EVENT: Consumer_Message_data_listener_exec_i::on_one_data: received message <App2> Hello World.  Test message from Provider <12>
[LM_DEBUG] - 23:45:17.183886 - MessageDispatcher::execute_message - Message executed
[LM_DEBUG] - 23:45:17.738411 - MessageDispatcher::execute_message - Deadline_type = DLT_NONE. Execute CMObject.
CONSUMER EVENT: Consumer_Message_data_listener_exec_i::on_one_data: received message <App1> Hello World.  Test message from Provider <0>
[LM_DEBUG] - 23:45:17.739916 - MessageDispatcher::execute_message - Message executed
[LM_DEBUG] - 23:45:18.180700 - MessageDispatcher::execute_message - Deadline_type = DLT_NONE. Execute CMObject.
CONSUMER EVENT: Consumer_Message_data_listener_exec_i::on_one_data: received message <App2> Hello World.  Test message from Provider <13>
[LM_DEBUG] - 23:45:18.182083 - MessageDispatcher::execute_message - Message executed
[LM_DEBUG] - 23:45:18.706913 - MessageDispatcher::execute_message - Deadline_type = DLT_NONE. Execute CMObject.
CONSUMER EVENT: Consumer_Message_data_listener_exec_i::on_one_data: received message <App1> Hello World.  Test message from Provider <1>
[LM_DEBUG] - 23:45:18.708378 - MessageDispatcher::execute_message - Message executed
[LM_DEBUG] - 23:45:19.180884 - MessageDispatcher::execute_message - Deadline_type = DLT_NONE. Execute CMObject.
CONSUMER EVENT: Consumer_Message_data_listener_exec_i::on_one_data: received message <App2> Hello World.  Test message from Provider <14>
[LM_DEBUG] - 23:45:19.183357 - MessageDispatcher::execute_message - Message executed
[LM_DEBUG] - 23:45:19.706862 - MessageDispatcher::execute_message - Deadline_type = DLT_NONE. Execute CMObject.
CONSUMER EVENT: Consumer_Message_data_listener_exec_i::on_one_data: received message <App1> Hello World.  Test message from Provider <2>
```

(b) App-2 log message snippet which shows that App-2 receives messages published by both itself and App-1 since it has higher security label than App-1

Fig. 6: Log message snippets for App-1 and App-2

the protocol and its average and maximum throughput were calculated and compared to native IPv6. Since ST is built on top of IPV6, each ST packet on the network incurs the same header overhead as an IPv6 packet.

Additionally, each message includes a ST-specific header containing information about the sending actor, the flow, the security labels, etc. This ST header is a minimum of 34 bytes for the smallest security label, averages around 60 bytes for most security label lengths, and has a maximum of 1,052 bytes for the longest security labels. Since a ST message can be up to 8 Kilobytes and an Ethernet packet is at most 1500 bytes, only one transmitted packet of each message will have the ST header. Since our test network on which we run IPv6 and ST is a tunneled 6-to-4 network, we must add the IPv4 header length to our total packet header. Finally, we choose UDP as our transport protocol for both IPv6 and the underlying ST protocol, so UDP's 8 byte header must be taken into account. The network utilization calculations are presented in Table II. These calculations are based on experiments executed on a private testbed of nodes connected to each other through a gigabit Ethernet switch.

TABLE II: Network Utilization Calculations for IPv6 and for Secure Transport tunneled through an IPv4 network.

| | Message Utilization (8192B) |
|---|---|
| IPV6 | 0.927956502 |
| Minimum ST Header | 0.924396299 |
| Average ST Header | 0.921692169 |
| Maximum ST Header | 0.820348488 |

## VI. Related Work

The OMG DDS specification, as currently specified, lacks an extensive security specification and therefore all DDS implementations lack a well-established security model. However, RTI [?] and PrismTech [?] have combined their efforts to put forward a DDS security proposal [?]. This proposal focuses on providing fine-grained, data-centric security by providing (1) access control per DDS topic, (2) read/write permissions for related DDS entities, and (3) field-specific permissions, which allow different fields of a particular topic to have different permissions.

Even though this proposal is critical to the advancement of the state of the art in DDS, however, a key issue that we have identified is that it relies on applications to correctly configure the security policies required, and the middleware to correctly enforce them. This requirement is fundamentally incompatible with a threat model where applications cannot be trusted. Furthermore, unlike our approach, this proposal is directed towards a solution that involves the entire DDS middleware to be part of the security model and therefore the TCB. Since the TCB is assumed to be trusted, care should be taken to keep its size small. Thus, maintaining the entire DDS middleware in the TCB is infeasible.

Similarly, prior efforts [?], [?] also require the entire DDS middleware to be part of the TCB. In [?], the authors present an approach in which Authentication and Authorization (AA) is embedded in the discovery mechanism which is implemented as a special network application located within the DDS Real-time Pub/Sub (RTPS) layer. This approach differs from ours since we place the centralized repository, used for discovery, outside of the DDS middleware. The work presented in [**?**], focuses on integrating Role-based Access Control (RBAC) with pub/sub communications. RBAC is similar to MLS in that the different roles (security labels) provide principals (actors) with different services (access restrictions). However, RBAC allows for the roles, in our case security labels, of a principal to change during the lifetime of the application, something that we do not support as we consider such flexibility as a potential source of attack.

In [**?**], the authors present the notion of microguards for data access restrictions. Microguards are distinct from the pub/sub middleware and are placed on domain boundaries to facilitate information sanitization via transitioning policies between any two domains. The authors argue that these microguards can be configured to perform checks that realize MLS policies. A limitation with guards is that they tend to be very specific to a data type and do not support generic data communications. Also, in their network configuration, the guards sit between networks or systems and connects them. They are not part of an actual system so it is possible to use a different communication path to bypass the security enforced by these guards. Such bypassing is not feasible in our solution. Furthermore, in our approach, all of the MLS-related label checks are done by a single entity that provides the discovery service rather than by distributing the task among multiple guards which introduces unnecessary complications and leads to an increase in the number of possible points of attack.

The Component Information Flow (CIF) framework presented in [**?**] provides a way of achieving data integrity and confidentiality during both intra- and inter-component communication. Labels are assigned to component ports at design-time via a policy file. However, that work has some limitations with the type of systems considered here because it supports label checking only at design and compile time, whereas we allow participants to arrive and leave dynamically.

## VII. Conclusions

This paper presented our work towards establishing secure publish/subscribe communication between applications with varying security classifications levels. These requirements were motivated by the open nature of distributed systems, such as the satellite clusters, in which applications from different vendors are likely to be hosted on the shared resources of the satellite cluster. Consequently, strict isolation among the traffic belonging to different applications and assuring communications only between entities that satisfy the MLS policies is needed. To that end we presented (1) Secure Transport, which is our transport mechanism that supports Multi-Label Security (MLS) labels and MLS policies to represent and validate security classifications, and (2) a secure discovery mechanism that is capable of establishing Secure Transport flows between DDS-based publish/subscribe applications that have matching security classifications. The use of a transport-level security mechanism ensures that no invasive changes need to be made to standard, application-level middleware, such as DDS. Our experimental evaluations indicate that our system provides the necessary information partitioning and that the extra overhead of the ST is negligible.

**Acknowledgments:** This work was supported by the DARPA System F6 Program under contract NNA11AC08C and USAF/AFRL under Cooperative Agreement FA8750-13-2-0050. 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 DARPA or USAF/AFRL. The authors thank Olin Sibert of Oxford Systems, Graham O'Neil of Kinsey Technical Services and all the team members of our project for their invaluable input and contributions to this effort.