Towards a Publish/Subscribe-based Open Policy Framework for Proactive Overlay Software Defined Networking

Akram Hakiri*, Pascal Berthou*, Prithviraj Patil† and Aniruddha Gokhale†
*Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France.
†ISIS, Dept of EECS, Vanderbilt University, Nashville, TN, USA.
Email: {hakiri, berthou}@laas.fr and {prithviraj.p.patil, a.gokhale}@vanderbilt.edu

***Abstract*—Proactive overlay software defined networking (SDN) aim to overcome the limitations with reactive hop-by-hop approaches for large-scale networks. However, the stateful nature of XMPP used by proactive overlay SDN approaches result in scalability issues. To address these concerns, this paper proposes the use of data-centric publish/subscribe paradigm, such as the OMG Data Distribution Service (DDS), for proactive overlay SDNs. To that end this paper makes two contributions. First, it presents a reference architecture called DDSFlex that can be used to build open and vendor-neutral, DDS-based southbound interfaces for proactive overlay SDN. Second, it describes a solution for realizing a SDN controller that operates in a distributed manner illustrating the messaging protocol it uses. Details of existing implementation of our approach are described.**

***Index Terms*—Software Defined Networking, Network Virtualization, DDS publish/Subscribe, OpenFlow.**

I. Introduction

Software-Defined Networking (SDN) [10] has emerged as a new intelligent architecture for network programmability, where the control plane logic is decoupled from the forwarding plane. The control plane embeds all the intelligence and maintains a network-wide view of the data path elements and links that connect them, which enables it to perform network management functions. The SDN community has adopted a number of northbound interfaces (i.e., between the control plane and applications) that provide higher level abstractions to program various network-level services and applications at the control plane. For the southbound interface (i.e., between the control plane and network devices), the OpenFlow standard [15] has emerged as the dominant technology.

Key application areas of SDN include (i) virtual home-gateways to improve service delivery and home energy management, (ii) multi-tenancy to enable network slicing in data-center interconnects, and (iii) traffic-engineering for greater control, flexibility and manageability of Internet Exchange Points (IXP) to facilitate the interconnection between service providers.

Proactive overlay SDNs are a variant of the SDN technology developed for large networks to address the scalability problems evident in the original reactive, hop-by-hop SDN networks [9], [14]. The overlay property stems from the use of overlay networks with multiple tunnels to create network slices whose endpoints terminate in a variety of entities, such as virtual switches or physical edge routers thereby not needing to contact any of the intermediate core routers and overcoming the problems with the hop-by-hop approach. The ability to pre-populate flows over these overlays provides the proactive capability.

The challenge posed by the proactive overlay approach is in deciding the technique to use in the control plane of SDN to program the endpoints of the tunnels that reside in the forwarding plane. One approach is to leverage the existing OpenFlow API [15], however, this API is too low-level and lacks intuitive abstractions that can support compositional semantics, which in turn forces developers to use error-prone techniques to link terms in the "match-action" rules [16]. These actions may lead to conflicting behaviors, policies and representations [5], which makes it difficult to design high-level abstractions to build reusable applications.

To overcome these issues, some vendors supporting the proactive overlay approach have adopted higher-level abstractions at the control plane, such as the Extensible Messaging and Presence Protocol (XMPP) [9]. However, we believe that there exist two limitations with this approach. First, XMPP is a wire protocol and is agnostic to the semantics of the data being transferred. Second, XMPP is a stateful protocol, which may be a source of scalability issues.

To address these concerns, we propose a middleware solution that provides the structure and semantics to build a common, open, vendor-neutral platform and computation-independent services. In particular, we propose using a publish/subscribe (pub/sub) middleware [4] since pub/sub supports an overlay network model natively and its broker-less model is well-aligned with proactive, overlay SDNs.

In our case a pub/sub middleware that supports a data-centric information model is most relevant since it has the expressiveness power to describe the states of network elements (e.g., creating route instances, exchanging routes and VPN membership information, managing link status for connection and disconnection) as well as for analytics to correlate, visualize and report statistics. From among the different pub/sub middleware alternatives, we surmise that the OMG Data Distribution Service (DDS) [18] provides the best option since it provides a high-level abstraction model that can be used to describe the information exchanged in the OpenFlow-based overlay virtual network, and its support for various quality of service (QoS) policies can be leveraged in enabling a proactive, overlay SDN.

This paper makes the following two contributions:

- We present the DDSFlex middleware, which is an extensible southbound protocol based on OMG DDS. DDSFlex is designed to exchange abstract policy between the network controller and a set of SDN switches capable of rendering that policy (e.g., packet forwarding, QoS class, MPLS labels, GRE tunnels, VXLAN ID, etc). DDSFlex supports a proactive model by using overlay tunnels to virtualize or "slice" the network – in contrast to the current, rigid reactive network – which enables it to support a combination of fine-grained flows in virtual edge devices and coarse-grained flows in the physical underlay core network devices, thereby representing the observable and controllable state of SDN network elements.
- We present the messaging protocol used by DDSFlex and describe the current implementation of our middleware-based solution to proactive overlay SDN.

The remainder of this paper is organized as a follows: Related work is described in Section II. The architecture of the DDSFlex is described in Section III. Section IV describes the messages exchanged and the information model of DDSFlex. The conclusions and learned lessons are described in Section V.

II. Related Work

This section compares related work to our proposed work on DDSFlex. We focus primarily of related work on middleware and related concepts for SDN.

SDN does not clearly identify how middleware platforms would interconnect applications to SDN network and provide the freedom to developers to define the way they would use it in SDN. For example, authors in [22] implemented the DISCO framework as an east-west interface to coordinate federated SDN controllers using RabbitMQ, which is an open source messaging broker based on the Advanced Message Queuing Protocol (AMQP) [26]. AMQP defines both wire protocol and protocol model that specifies the semantics for DISCO implementation. Another important aspect is that DISCO enables the broker to make routing decisions that are usually left to the application. However, DISCO can be considered as graph of nodes connected by links. In contrast, DDSFlex supports self-formed federation that reads and writes topics over the global data space. DDSFlex can operate as repository federation in which individual repositories can participate in a global federation in fully distributed manner. That is, DDSFlex serves as a mediator between controllers and the OpenFlow-capable switches, which makes it easier to enhance scalability and flexibility.

Many middleware implementations for content-based routing have been developed over the last decade [1]–[3]. The authors in [25] propose a content-based routing middleware over overlay networks. The authors in [27] [20] studied the feasibility of pub/sub content-based overlay design. Similarly, the work in [11] propose a pub/sub architecture for SDN, where the controller establishes line-rate content-based matching semantics to disseminate routing information to SDN-enabled switches. Nevertheless, the most fundamental problem of content-based routing systems is that they are not suitable for large-scale, widely-distributed SDN. We argue that data-centric QoS-enabled pub/sub systems enables loosely coupled, fully distributed and highly scalable communication that let them more suitable for widely distributed SDN.

Unlike these approaches, DDSFlex proposes an efficient matching of advertisement and subscription on pub/sub middleware to support flexible and programmable network, while enhancing the expressiveness of content-based routing with topic-based pub/sub supported by OMG DDS. Thus, with our reference architecture it is possible to configure the forwarding tables of switches directly through the middleware using a software control layer inside the SDN controller running in general-purpose external computer.

Recently some efforts are exploring the Extensible Messaging and Presence Protocol (XMPP) service as an alternative or complement to OpenFlow in hybrid SDN networks [13]. The XMPP middleware is used to distribute control plane and management plane information to end server that serves to enhance the communication between data centers in overlay network and physical devices in the underling network. The disadvantages of XMPP and its related technologies (i.e., roster, presence and routing functions) exist in different contexts. In particular, XMPP is considered to be a standard only for the wire protocol, i.e., XMPP is agnostic in relation to the data being transferred. XMPP is also stateful which makes it more difficult to scale because each server needs to know the entire state in order to serve a request. Typically, the XMPP clients and servers utilize the domain name system (DNS) to resolve a server's domain name into an address they can connect to. XMPP also demands centralized services to exchange messages between server-to-server federation, which makes it inefficient in duplicating messages when distributing them to multiple destination. This is where utilizing DDSFlex for message distribution is more beneficial then XMPP. DDSFlex uses built-in DDS discovery service to allow publishers and subscribers to dynamically and continuously discover each other without the need to contact any name servers.

The OpFlex control protocol is introduced in [24] to configure and monitor all connected devices. The OpFlex protocol is founded in the concepts of declarative policy driven system to control and program a large set of physical and virtual network devices. OpFlex is a request-response protocol based on JSON-RPC [17] where each component sends a request to query the information from its peer element. However, there are several disadvantages of RPC with respect to message passing, since it may incur severe penalty in performance due to marshaling/unmarshaling of messages (i.e., context switching increases scheduling costs) and may have to deal with added complexity in configuration for simple scenarios.

Unlike OpFlex, DDSFlex is able to dynamically program policy across even multi-vendor networks, which makes the infrastructure dynamically responsive to the needs of applications. DDSFlex is fully distributed so there is no single point of failure in the network during the communication. Additionally, DDSFlex supports reconfigurable DDS QoS policies which enables it to manage the use of the bandwidth, network and memory resources. Likewise, DDSFlex offers QoS policies to prioritize messages, guarantee QoS properties in the communication (bandwidth usage, delivery semantics, etc) and control many aspects of the reliability of the messages between fully distributed and loosely coupled participants. Furthermore, it provides very compact binary encoding for both protocol messages and data payload, and supports bounded use of resources over potentially intermittent links, making it feasible to reply to messages upon reconnection. That is, DDSFlex natively fulfills the requirements of the future proactive SDN communication which makes it the most suitable technology to satisfy the features they need, such as scalability, reliability, flexibility, security and real-time data.

III. DDSFlex Architecture

This section presents the DDSFlex architecture which extends SDN using DDS to promote the concept of proactive, virtual overlays for SDN, and provide an open, vendor-neutral and extensible policy framework to exchange abstract information between the SDN controller and the set of devices capable of rendering the policy. Figure 1 depicts the DDSFlex architecture. The rest of the section delves into the design rationale and details of the DDSFlex architecture.

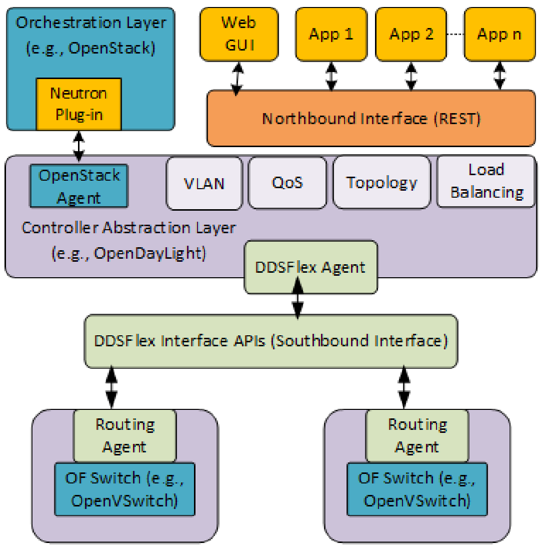

Fig. 1. DDSFlex Architecture

A. DDSFlex Design Rationale

Before delving into the details of the DDSFlex architecture, we provide the rationale for the architectural decisions we made.

Requirement 1: Need for an open, vendor-neutral, extensible solution –: Recall from Section I that we need a solution that (a) supports the proactive overlay SDN approach, and (b) provides an open, vendor-neutral approach that supports intuitive interfaces at a higher-level of abstraction without the scalability limitations of existing higher-level abstractions. It is also necessary to provide a dynamically extensible architecture where network devices can be introduced into the system without any disruption to the operational system. Moreover, it is important to provide the users with a generic policy framework that hides the heterogeneity in the underlying network devices. Thus, it is desirable to have a common interface and messaging capability between the control plane and forwarding plane where new devices can be seamlessly added. This motivates a middleware-based solution.

Requirement 2: Remain compliant with the SDN architecture –: The middleware solution must be designed in a way that is compliant with the decoupled architecture of SDN where the control plane is decoupled from the data plane. To that end, DDSFlex supports two key elements: a DDSFlex agent and a routing agent. DDSFlex relies on a policy management service that is understood by a physically centralized but logically distributed set of DDSFlex agents that exist in the control plane, and a routing agent that exists in the forwarding plane, which receives these policies and translates the high-level abstraction information to low-level configuration/commands understood by the specific switch. The DDSFlex agent at the control plane communicates directly with the controller (through the node processes) while the routing agent implements functionalities related to the forwarding plane. It comprises interfaces to communicate with the DDSFlex agent through the DDS Flex protocol to exchange messages with it as well as with the switches in the forwarding plane. Thus, this interface must be able to translate the messages received from the control plane to the matching rules understood at the switch's forwarding tables.

Requirement 3: Data-centric pub/sub with QoS –: In the context of our two-level architecture, two sub requirements arise. First, since the underlying network can be dynamically reconfigured (e.g., switch can be added), there is a need for the control plane to be notified whenever there is a change in the underlying physical network. Second, despite the potential for dynamic changes, it is important to provide the user with a flexible and higher-level of abstraction to program the system that can handle QoS issues and policies. This motivates the use of a data-centric pub/sub middleware provided by OMG DDS. Thus, in DDSFlex the information exchange is performed by the DDSFlex protocol that provides a built-in discovery service to match every participant (i.e., DDSFlex agent and routing agent). This way it is possible for the DDSFlex reference architecture to render services to vendor-neutral smart devices while simultaneously providing more flexibility and programmability to users. The application communication requirements (such as QoS, link connection to virtual switch, multi-path splitting to several switches) requires the high-level abstraction model to configure the forwarding tables of the virtual switches and to map the low-level configuration to the physical infrastructure.

B. Overview of OMG DDS

Since OMG Data Distribution Service (DDS) plays a major role in the DDSFlex architecture, we provide a brief overview of OMG DDS standard. The OMG DDS standard adopts a data-centric, topic-based publish/subscribe communication model. A topic is the atomic unit that can be shared between data publishers and subscribers. Topics are fully defined by their names and types, and define the data structure which publishers and subscribers write and read within a DDS Global Data Space. DDS provides flexibility and a modular structure by decoupling: (1) *location*, via anonymous publish/subscribe, (2) *redundancy*, by allowing any numbers of readers and writers, (3) *time*, by providing asynchronous, time-independent data distribution, (4) and *message flow*, by providing data-centric connection management.

Publishers and subscribers discover each other automatically and match whenever they have compatible topics and QoS policies. Topic samples are exchanged between peers within the global data space according to a contract established in the discovery phase. DDS can use multiple topic samples called instances that are differentiated by their associated unique key. Topic samples can be configured with a wide range of DDS QoS capabilities imposed by peers. The underlying DDS data dissemination is fully decentralized and adopts a peer-to-peer un-brokered service model, which eliminates single points of failure for brokers. For additional details on DDS, we refer the reader to our prior work on supporting DDS in wide area networks [8] or other published literature on DDS.

C. DDSFlex Agent: The Control Plane

Figure 2 shows the internal structure of the DDSFlex agent which comprises four types of nodes (these are essentially software processes): monitor node, policy node, configuration node and control node. The monitor node collects information related to the health of the system including faults, errors, etc. The policy node includes a policy decision point and policy enforcement point both connected to a policy data store. The configuration node communicates with the orchestration layer via REST APIs to the applications. It also communicates with other configuration nodes through distributed synchronization services provided by the built-in discovery mechanisms and with the policy node through a request-response model.

1) Control Node: The control node (illustrated in Figure 2) communicates with all other nodes since it is responsible for communicating the high-level abstraction model to the routing agent via the DDSFlex protocol (explained in Section IV. It receives configuration states from the configuration node using content-based filtered topics supported by DDS, which contain the forwarding rules it will send to the routing agent to install new rules in the forwarding plane.

Additionally, it exchanges network route information with other distributed control nodes and routing agents, and sends

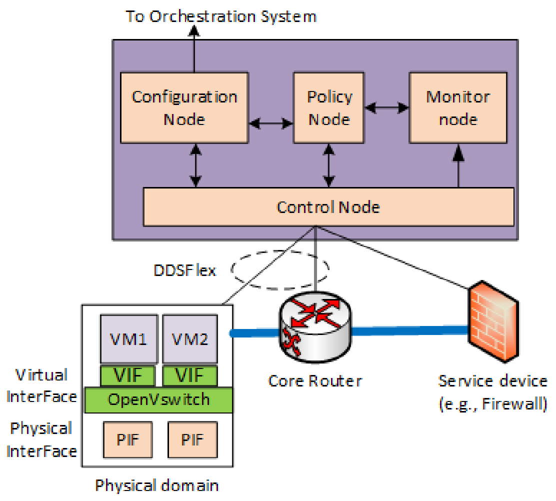

Fig. 2. Internal components of the DDSFlex Agent

forwarding policies into the policy database to enable intelligent decision making without resorting to the controller every time a new flow arrives. The control node holds a global snapshot of the network and the link states which helps it in performing the best route selection. Hence, the SDN controller would be able to configure overlay tunnels based on the high-level information provided by the configuration node and the policy management service inside the DDSFlex agent. At the forwarding plane, the routing agent will be able to collect route information provided by the classical route advertiser of the BGP protocol from the switches, or even VLAN tags and virtual routes inside a data center tenant. These low-level data are then published to the control plane and the orchestration layer through the DDSFlex agent.

2) Configuration Node: The configuration node communicates with the orchestration in the northbound interface via REST APIs. Examples of orchestration include cloud infrastructure, such as OpenStack, CloudNaaS or even any other cloud computing platform that is able to manage data center infrastructure (i.e., IaaS). REST interfaces could be used as well to install configuration states, such load balancing, at the high layers. Furthermore, the configuration node can communicate with the policy node to query the optimal policy to apply for the purpose of network function virtualization in the orchestration or even apply a specific policy decision to the network device through the control node.

3) Monitoring, Analysis and Troubleshooting: The DDSFlex agent performs flexible and robust monitoring built atop a DDS pub/sub model. It enables data dissemination to the monitor node that gathers statistics, faults, and errors from the underlying layers. Figure3 shows the internal structure of the monitor node. A monitor node supervises the network resources and communicates with applications using northbound REST interfaces. The REST APIs are used to query the statistics and analysis, and retrieve the operational states of the network provided by the DDSFlex agent. Also, the monitor node advertises the DDSFlex agent using the distributed management module, which in turn uses the built-in DDS discovery service (i.e, built-in topic entities) to collect data from the network.

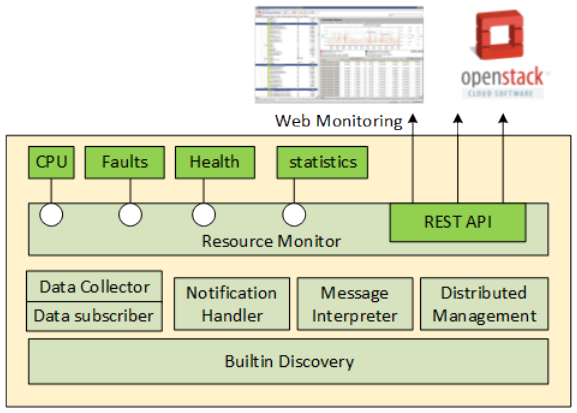

Fig. 3. Monitoring node

The monitor node uses the data collector to communicate with the policy management module through the content-filtered topic interface. That is, the monitoring service uses filter expressions to select data samples of interest using SQL like expression to store the information in the database. Moreover, the data collector subscribes to two kind of messages: asynchronous events received from the notification handler module for the purpose of reporting logs, statistics, and events and traces, and synchronous messages whereby a policy management handler can send a message to the resource mapping handler.

4) Policy control: A flexible policy abstraction is required to interact safely and scalably with BGP. Instead of supporting numerous policies, the abstraction policy layer should combine policies from multiple participants to generate a set of rules that apply the BGP routes without flooding the rules and flow tables. For example, since each autonomous system uses a BGP speaking router (also called BGP advertiser) at the forwarding plane and BGP server for exchanging routing information at the control plane, each current AS specifies a complex set policies to forward/drop/update/modify traffic.

The development of policy control raises many important aspects for how the management and services are supported and delivered by the distributed DDSFlex agents. It is also useful to express delegation of authority and conflict resolution when resources are shared as well as to help SDN controllers to decide how to setup and manage flows. Policy control functions ensure that the DDSFlex agent can continue to honor guaranteed service delivery to end-users. They also determine how network resources are allocated as well as how individual subscribers can take into account network flow control and application-oriented flow. To this end, the policy node in Figure 2 can infer policies from applications using northbound interfaces through the configuration node as well as from the southbound interfaces using the control node. The policy module provides a high-level abstraction to specify how packet forwarding/updating policies should be effected. The policy abstraction is built atop Pyretic [6] to express more flexible policies that resolve conflicts and by composing different policies into a single set of matching rules in the network devices.

Figure 4 depicts the internal structure of the policy management service inside the policy node. The configuration node uses the policy control functions to ensure that resources in the orchestration layer will be available to meet critical network needs. Furthermore, the DDSFlex agent could support some of packet classification from the DDS transport priority QoS policy. These policies can map the requirements of the applications into specific differentiated class of services (CoS) that enable action-matching directly. Examples of mapping DDS QoS policies into network packet classification were described in our prior work involving DDS over large-scale overlay network [7], [8]. The DDS transport_priority QoS policy is a hint to the infrastructure used to set the priority of the underlying transport used to send data in the DSCP field for DiffServ.

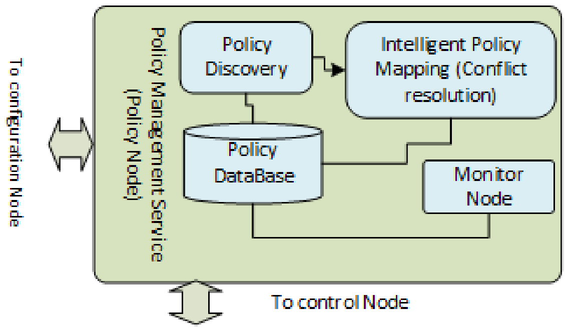
To configuration Node
Policy Management Service (Policy Node)
Policy Discovery
Intelligent Policy Mapping (Conflict resolution)
Policy DataBase
Monitor Node
To control Node

Fig. 4. Policy Service

D. Routing Agent: The Data Plane

As depicted in Figure 5, the routing agent is a user space process available on each instance of an OpenFlow capable switch and acts as a local translator at the forwarding plane. It is responsible for exchanging control states (e.g., MPLS label, BGP Network Layer Reachability Information (NLRI), bandwidth usage, etc.) between the DDSFlex Agent in the control plane and the network devices in forwarding plane through the DDSFlex protocol.

The routing agent receives high-level configuration information from the DDSFlex agents and translates them to a low-level details (i.e., QoS, VLAN tags, etc.) understood by the switches in the overlay. It installs the forwarding states in the group tables within the switches. Moreover, it carries and reports the analytics, heath states, errors and statistics from the forwarding plane to the distributed DDSFlex Agents. Additionally, the routing agent interfaces with the orchestration layer to provide network function virtualization. Thus, the routing functions virtualized within in the orchestration layer are provided to the router through the routing agent.

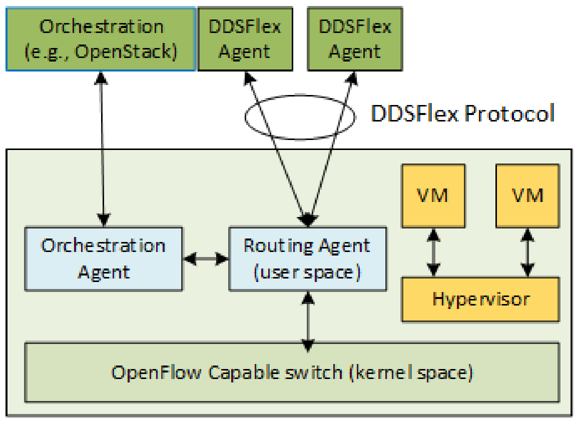
Orchestration (e.g., OpenStack)
DDSFlex Agent
DDSFlex Agent
DDSFlex Protocol
VM
VM
Orchestration Agent
Routing Agent (user space)
Hypervisor
OpenFlow Capable switch (kernel space)

Fig. 5. Internal view of the routing agent inside OpenFlow capable switch

E. Securing Flow Forwarding in DDSFlex

As DDSFlex would interconnect hundreds of SDN controllers and devices at different scales where data are shared between participants among shared data spaces, security is one of the leading challenges in SDN networks because its programmable aspect presents a complex set of problems to cope with. Additionally, as security is minimally specified in SDN, the increasing number of DDoS and malware attacks, spam, and phishing activities will change the dynamics around securing SDN infrastructures. Therefore, data security will involve more sophisticated encryption and authentication mechanisms to prevent hackers and recover packets from failure. Also, DDSFlex should ensure that all participants are authenticated and have the necessary credentials before entering the DDS domain. It should also ensure some level of security to provide data integrity, pedigree, confidentiality, and non-repudiation. For example, it should be able to provide a fine-grained group-based access control in a DDS domain, e.g., role-based and policy-based access control. We describe how DDSFlex can address these concerns.

To that end, DDSFlex can leverage the Security Model (SM) described by the OMG DDS specification [19] to ensure simple access and interoperable security policies without compromising the flexibility scalability, performance, QoS-awareness, and robustness offered by the DDS platform. Additionally, DDSFlex can use the DDS security model and its Service Plugin Interface (SPI) to build a fine-grained secure system that grants permissions to DDS domains, Topic or even data object instances within the Topic. For example, DDSFlex should ensure that messages exchanged with the routing agent will not be altered by DDoS and malware attacks. In addition, because the DDSFlex reference architecture is fully distributed, the communication between DDSFlex agents will be exposed to multiple vulnerabilities that would change the behavior of the system. Therefore, subjecting our reference architecture by a high-level DDS native security policies will improve the integrity and the security of the system.

F. Benefits accrued using DDS

Multiple benefits are accrued using DDS to realize DDSFlex, which are described below. First, the asynchronous and anonymous pub/sub communication supported by DDS is ideal for DDSFlex agent anonymity. In particular, the DDSFlex agent does not have to reply on the existence of other DDSFlex agents to act or setup communication, so agent modules can be created dynamically at any time and at any node without using a specific order; that is, DDS is suitable for intermittent connectivity scenario, because its aspect of plug & play of system nodes and its dynamic and automatic discovery of the nodes/services.

Second, DDS provides a local caching mechanism specified by the discovery protocol, the Real Time Publish Subscribe. In particular, when publishers push topic samples to the remote peers, the latter maintain a copy of them in their local cache. Thus, subscribers can continue performing query operations from their local cache without generating any network traffic, mainly for efficient use of network resources. Additionally, DDS supports topic-based filtering capabilities to optimize the use of resources, such as bandwidth, by sending the most relevant data samples to subscribers. For example, DDS publishers do not send topic instances that not match some filtering criteria imposed by the subscriber.

Third, DDS provides built-in data isolation, known as domain and partitions, to operate in a scalable fashion. In particular, DDSFlex agents belonging to different partitions can easily be separated from one another in their isolated domains, or even allow a coordinator agent to communicate within different slices and/or domains. Additionally, DDS can check whether DDSFlex agents are still alive and exchanging DDS topics within the same global data space and partitions. For example, DDS Liveliness QoS policy can be used to monitor the states of DDSFlex agents and discover the new agents joining the domain, or even can check whether discovery messages are being sent periodically between agents. This is useful in checking if the physical devices are alive or dead.

Finally, DDS defines a set of scheduling policies (*e.g.*, latency budgets), *timeliness policies* (*e.g.*, time-based filters to control data delivery rate), temporal policies to determine the rate at which periodic data is refreshed (*e.g.*, deadline between data samples), priority policies (*e.g.*, the transportation of the messages can be regulated with the respect of their importance and priorities, and other policies that affect how data is treated once in transit with respect to its reliability, urgency, importance, and durability.

IV. DDSFlex Messaging Protocol

In this section we will describe the messaging protocol and message types supported by DDSFlex to enable flow forwarding over the proactive overlay SDN. Three kinds of messages are supported and described in this session: discover, configuration, and monitoring messages.

DDSFlex uses the programming model and API of the OMG DDS specification to describe different policies and information exchange between entities. Thus, for every message kind in DDSFlex, we use a DDS Topic to define the message type. Among the three message kinds, the configuration messages will receive the highest scheduling priority to ensure that control flow is processed in the controller before any other flow. It also will receive the highest DDS QoS transport priority to ensure that data will be protected against dropping when it traverses the network devices.

A. Discovery of Peers

When setting up flow forwarding in the proactive overlay SDN approach, it is important to identify the tunnel endpoints, and thereby the DDSFlex entities, such as DDSFlex agents and routing agents. This requires discovery of such entities.

DDSFlex relies on the underlying discovery mechanism, such as OMG DDS' Real-Time Publish-Subscribe (RTPS) protocol, to enable : i) different participants to learn about each other (i.e., by sending participant declaration messages, also known as participant DATA sub-messages); and ii) DataWriters and DataReaders exchange information (i.e., such as QoS, type types, etc.) to match each other (i.e., by sending publication/subscription declarations in DATA messages also known as publication DATAs and subscription DATAs). For this we will need a session management protocol that enhances the DDS discovery process, which is part of our future work. Discovery messages are sent periodically (i.e., regulated by a heartbeat) to check the liveliness of different DDSFlex entities whereby there is no need to send ping messages to check the reachability of the entities.

Listing1 shows a snapshot of the discovery topic exchanged during the discovery phase between DDSFlex agents and routing agents. This Listing depicts some information that would be used during the discover process: for example, the "sender" would describe the IP address (also it may be MAC address in the context of layer 2 participant) of publisher, the "sender_ID" would be a "VLAN ID" of a participant or any other membership information that can uniquely identify a participant within its group. Also, the discovery process uses DDS QoS policies regarding the DDS specification. For example, "LivelinessQoS" would ensure automatic discovery of entities. Moreover, the discovery process would introduce a session management protocol to ensure that communication will established end-to-end. To that end, we introduce a novel in-bound session management protocol. It creates a DDS session between remote entities. The session management is similar to DDS over SIP as described in our prior work [8], but it differs from it in couple of things: i) DDS/SIP session is out-of-the-bound protocol so control traffic and data traffic are sent over two different channels, where session management in this case is "in-the-bound" whereby control flow and data flow are in the same channel; ii) the DDS/SIP protocol used a centralized server (also called registrar to maintain a local a care of address), which is in contrast to this approach where there is not single server.

Listing 1. Built-in participant Discovery

```
<?xml version="1.0"?>
<-- Discovery topic -->
<struct name="discovery">
 <member name="sender" type="string"
key="true"/>
 <member name="sender_ID" type="string" />
 <member name="receiver" type="string"/>

 <!-- DDS QoS policies ... -->
 <member name="LivelinessQoS" type="string"/>
 <member name="DurabilityQoS" type="string"/>
 <member name="TopicDataQoS" type="string"/>
 <member name="GroupDataQoS" type="string"/>
 <!-- more DDS QoS policies ... -->
<!-- Session Management -->
 <member name="scope_announce" type="sequence">
 <member name="register_context" type="sequence">
 <member name="session" type="sequence"
key="true"/>

</struct>
```

In listing1, the field "scope_announce" describes the announcement of the session from participant to all other participants. The "scope_announce" is a sequence of fields that would include the forwarding path, the session heartbeat, the PDP data, and more. Please note that session management protocol is not in the scope of this paper.

B. Network Configuration Operation

Once the peers are discovered it is important for the DDSFlex agent to configure all the identified routing agents with information pertaining to the flow. Listing 2 illustrates a network configuration topic that would be used by the DDSFlex agent to populate the Routing Information Base (RIB) of the routing agent.

Listing 2. Configure BGP Path

```
<?xml version="1.0"?>
<struct name="bgp_advertise">
 <member name="sender" type="string"
key="true" />
 <member name="receiver" type="string"/>
 <member name="sender_id" type="long"/>
 <member name= "node" type="string" />
 <member name="instance-id" type="long">
 <--! BGP Network Layer Reachability
 Information (NLRI) -->
 <member name="nlri" type ="string">
 <!-- Label for label switch routers (LSRs) -->
 <member name="label" type="long">
</struct>
```

C. Monitoring messages

With any networking technology, there is a need for network management to deal with failures and collect different kinds of statistics. DDSFlex introduces new possibilities for network management and monitoring capabilities to improve performance, reduce the bottlenecks in the network, and enable debugging and troubleshooting of the control traffic. DDSFlex supports this requirement through the monitoring message. Listing 3 shows an example of topic data that would be used by DDSFlex to query the state of an SDN switch through the routing agent. This topic message provides the required information about the fault, statistics, errors, and health of the network device. This message can include the information about the sender of that request (e.g., IP address, VLAN tag), the entity which sent the report and other information including statistics, faults, and errors, etc.

Listing 3. Query Network Statistics

```
<?xml version="1.0"?>
<struct name="report_sate">
 <member name="sender" type="string"
key="true" />
 <member name="sender_id" type="long"/>
 <member name="receiver" type="string"/>
 <member name= "node" type="string" />
 <member name="statistics" type="" >
 <member name="faults" type="">
 <member name="health" type="">
</struct>
```

D. Implementation Details

Prototyping our ideas was not an easy task since it is not readily possible to install OMG DDS in today's Openflow-enabled switches. Yet we had to validate our ideas. Therefore, we decided to implement our ideas inside a SDN emulation environment. To that end we chose the Mininet SDN emulation environment [12] and the RTI Connext [23] as DDS distribution to demonstrate how DDSflex agent can be incorporated in SDN and how it can be used for the dynamic controller and switch interaction.

Mininet is a network emulator available in the Linux OS used to rapidly realize large networks inside a single machine, such as a laptop. It uses lightweight OS virtualization features, such as processes and network namespaces, to make a single system look like a complete network. It is within this emulation environment that Mininet provides a set of tools to experiment with SDN capabilities and entities, such as the controller, switches and hosts. It uses the Openflow API for the southbound interface for communication between the controller and switches. Mininet also comes preinstalled with Open vSwitch (OVS) [21].

We implemented two different ways of incorporating DDS-flex in the SDN controller and switches. In the first approach, we use the tunneling approach to preserve most of the SDN Mininet architecture. Thus, our attempt is to combine both the protocols (i.e., Openflow and DDS) to take advantage of their respective strengths. We use OMG DDS for communication but use Openflow message structure to format the flow table information. This is done by converting various Openflow messages into OMG DDS topics. We use POX for the controller and Open vSwitch for switch in the Mininet environment. For this, we modified the POX controller implementation to install a DDS publisher with a data writer. We also modified the Mininet switch initialization process by adding a listening endpoint and starting a DDS subscriber listening on that endpoint. Openflow messages are thus encapsulated within OMG DDS messages on the publisher side, and on the subscriber side and decapsulated to retrieve the topic type, which is the Openflow message structure, which is understandable by the switches.

The tunneling approach may incur overhead due to the level of indirection. Thus, in our second approach we used DDS for both communication and exchange of flow table information thereby replacing Openflow with DDS messaging altogether. Note, however, that this solution will be incompatible with Openflow-only switches. This implementation is more involved since it requires substantial modifications and hence this approach is still a work in progress.

V. Conclusion

Software-Defined Networking (SDN) is an emerging architecture that decouples the network control plane from the forwarding plane, which makes it possible to program the network control plane independently of the forwarding plane while abstracting the underlying infrastructure comprising various kinds of network devices. SDN makes it cost effective to deploy and manage applications that are dynamic and have high bandwidth requirements. SDN supports a range of network management approaches from reactive, hop-by-hop to proactive, overlay SDNs. In the context of the proactive overlay approach, it is desirous to support APIs that are intuitive and provide higher levels of abstraction to program the control plan while also supporting scalability, performance, reliability and security in a manner that is open and vendor-neutral.

To address these requirements, this paper presented a middleware solution called DDSFlex that uses the OMG DDS data-centric publish/subscribe paradigm. The solution comprises a two-level architecture that aligns with the decoupled architecture of SDN. The messaging protocol of DDSFlex was presented along with three different use cases that can benefit from using DDSFlex.

We have currently prototyped the ideas presented in this paper in an emulation environment using two different approaches. A rigorous validation of the framework capabilities comparing its performance and resilience properties with existing solutions is needed. A further dimension of research involves supporting DDSFlex in a hybrid SDN scenario: one that employs both reactive hop-by-hop at the edge, while proactive overlays for the core network. We believe that further architectural changes will need to be made in how end-to-end tunnels are created. A possibility is by using multiple instances of DDSFlex that would communicate with each other and will require east-west interfaces. These dimensions of research are part of our ongoing research.

Acknowledgments

This work was partially funded by the French National Research Agency (ANR) and the French Defense Agency (DGA) under the project ANR DGA ADN (ANR-13-ASTR-0024)

References

[1] Roberto Baldoni, Carlo Marchetti, Antonino Virgillito, and Roman Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, ICDCS '05, pages 437–446, 2005.

[2] Jorge A. Briones, Boris Koldehofe, and Kurt Rothermel. SPINE : Adaptive Publish/Subscribe for Wireless Mesh Networks. *Studia Informatika Universalis*, 7(3):320–353, Oktober 2009.

[3] Jaeyoung Choi, Jinyoung Han, Eunsang Cho, T. Kwon, and Yanghee Choi. A survey on content-oriented networking for efficient content delivery. *Communications Magazine, IEEE*, 49(3):121–127, March 2011.

[4] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, jun 2003.

[5] Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. Hierarchical policies for software defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 37–42, 2012.

[6] N. Foster, A. Guha, M. Reitblatt, A. Story, M.J. Freedman, N.P. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison. Languages for software-defined networks. *IEEE Comm Mag*, 51(2):128–134, February 2013.

[7] Akram Hakiri, Pascal Berthou, Aniruddha Gokhale, Douglas C. Schmidt, and Thierry Gayraud. Supporting end-to-end quality of service properties in OMG data distribution service publish/subscribe middleware over wide area networks. *Journal of Systems and Software*, 86(10):2574 – 2593, 2013.

[8] Akram Hakiri, Pascal Berthou, Aniruddha Gokhale, Douglas C. Schmidt, and Gayraud Thierry. Supporting sip-based end-to-end data distribution service qos in WANs. *Journal of Systems and Software*, (0):–, 2014.

[9] Juniper Networks, Inc. Proactive Overlay versus Reactive Hop-by-Hop: Juniper's motivations for the Contrail architecture. Technical report, Juniper Networks, Inc, 2013.

[10] Hyojoon Kim and N. Feamster. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119, 2013.

[11] Boris Koldehofe, Frank Dürr, Muhammad Adnan Tariq, and Kurt Rothermel. The power of software-defined networking: Line-rate content-based routing using openflow. In *Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing*, MW4NG '12, pages 3:1–3:6, 2012.

[12] Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[13] P. Marques, L. Fang, P. Pan, A. Shukla, M. Napierala, and N. Bitar. Bgp-signed end-system ip/vpn, October 2013.

[14] Diogo Mattos, Lyno Ferraz, Luís Costa, and Otto Duarte. Virtual network performance evaluation for future internet architectures. *Journal of Emerging Technologies in Web Intelligence*, 4(4), 2012.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, and J. Rexford. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, pages 69–74, 2008.

[16] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software-defined networks. In *10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 1–14, Berkeley, CA, USA, 2013. USENIX Association.

[17] Matt Morley. JSON-RPC 2.0 Specification, Mar 2010.

[18] Object Management Group. Data-Distribution Service for Real-Time Systems - Version 1.2. http://portals.omg.org/dds/, January 2007.

[19] Object Management Group, Inc. (OMG). *DDS Security Extensions RFP Proposal*, mars/2014-02-03 edition, February 2014.

[20] M. Onus and A.W. Richa. Minimum maximum-degree publish-subscribe overlay network design. *Networking, IEEE/ACM Transactions on*, 19(5):1331–1343, Oct 2011.

[21] Justin Pettit, Jesse Gross, Ben Pfaff, Martin Casado, and Simon Crosby. Virtual Switching in an Era of Advanced Edges. In *2nd Workshop on Data Center–Converged and Virtual Ethernet Switching (DC-CAVES)*, 2010.

[22] Kevin Phemius, Mathieu Bouet, and Jeremie Leguay. Disco: Distributed multi-domain sdn controllers. *CoRR*, abs/1308.6138, 2013.

[23] Real-Time Innovations (RTI). Rti connext dds software. http://www.rti.com/products/, January 2014.

[24] M. Smith, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher. Opflex control protocol, April 2014.

[25] Naweed Tajuddin, Balasubramaneyam Maniymaran, and Hans-Arno Jacobsen. Minimal broker overlay design for content-based publish/subscribe systems. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13, pages 25–39, 2013.

[26] S. Vinoski. Advanced message queuing protocol. *Internet Computing, IEEE*, 10(6):87–89, Nov 2006.

[27] Yaxiong Zhao and Jie Wu. On the construction of the minimum cost content-based publish/subscribe overlays. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011 8th Annual IEEE Com Soc Conf on*, pages 476–484, June 2011.