# The Role of Data Distribution Service in Failure-aware SDN Controllers

Alejandro Llorens-Carrodeguas[(1)], Cristina Cervelló-Pastor[(1)], Irian Leyva-Pupo[(1)],
Juan Manuel López-Soler[(2)], Jorge Navarro-Ortiz[(2)]
[(1)]{alejandro.llorens, cristina, irian.leyva}@entel.upc.edu
[(2)]{juanma, jorgenavarro}@ugr.es
[(1)]Universitat Politècnica de Catalunya (UPC), Barcelona, Spain.
[(2)]University of Granade (UGR), Granade, Spain

***Abstract*—The necessity of a flexible, scalable, faster and programmable network to offer new services is challenging to telecommunication operators. These characteristics along with robustness and availability are essential in order to guarantee a Quality of Service (QoS) and Quality of Experience (QoE) in the users. Software Defined Networking (SDN) has emerged as a natural solution to this situation as it enables network programmability. The use of Data Distribution Service (DDS) is studied in order to achieve robust and efficient SDN controller federations. In this paper we deploy a 5G scenario with SDN controllers and analyze the roll of DDS to improve the controllers performance. Illustrative results demonstrate an enhanced performance in determining when a controller fails.**

## I. Introduction

Nowadays, there is a tremendous growth of costumers' demands due to the appearance of new types of applications and services such as augmented and virtual reality and so on. This situation along with the existence of new environments like Internet of Things (IoT), Cloud Computing and Big Data are driven to take into account a new network paradigm.

In order to satisfy the new telecommunication service requirements, the traditional networks must not be used because of their lack of programmable environment and the strong coupling between hardware and software in their physical devices.

The advantages of a decoupled data and control plane deliver new means and methods to instantiate networks and services, reducing expenses and boosting performance. These characteristics make Software Defined Networking (SDN) a powerful tool to develop, deploy and operate networks. The resilience of a core network can be improved by means of a set of distributed controllers instead of a single controller as the number and size of production networks deploying Openflow increases, the network feasibility decreases. However, a distributed core network implies a higher level of synchrony between their elements which means a higher traffic in the network.

Data Distribution Service (DDS) is a OMG's standard middleware for distributed real-time applications which is based on the Publisher/Subscriber paradigm. It simplifies the application development, deployment and maintenance and provides fast and predictable distribution of time-critical data over a variety of transport networks. It delivers large amount of data with microsecond performance and granular Quality of Service (QoS) control using a distributed cache, referred to as data-space. DDS decouples in time and location the data producers and consumers. The communication is established using Topics, which are data streams of the same data type. By means of DataWriters and DataReaders, publishers and subscribers can publish and subscribe DDS samples into and from the streams in the common data-space [1].

In this work, we focus on improving the scalability, resilience and reliability of an SDN network using DDS as the first step to take into account in 5G Networks. Our approach avoids single-point controller failure, and consequently not only improves the network resilience, but also network scalability because multiple collaborative controllers can share network information among each other. At the same time, we enhance the performance of our design by means of applying some parameters of QoS in the DDS. The following sections contextualizes the addressed problem and present related studies. In Section III, we explain our design. It includes details about the integration of DDS with OpenDaylight SDN controllers. Section IV provides some insights about the implementation and evaluation of our design. Finally, conclusion are given in Section V.

## II. Related Work

Several attempts have been done to distribute SDN controllers and exchange network information among each other.

In [2], the authors propose a distributed event-based control plane for Openflow called HyperFlow, which allows network operators to deploy any number of controllers in their network. HyperFlow provides scalability while keeping network control logically centralized. Each controller publishes events related to the state of the system, while other controllers replay all the published events to reconstruct the state. HyperFlow uses the Publish/Subscribe paradigm to facilitate cross-controller communication by means of WheelFS, which is a distributed file system designed to offer flexible wide-area storage for distributed applications.

Similarly, Koponen et al. in [3] propose a distributed system which runs on a cluster of one or more physical servers. These servers may run multiple Onix instances that are responsible for disseminating network state to other instances within the cluster. The network state is stored in a data structure called Network Information Base (NIB). The NIB is a graph of all network entities within a network topology. ONIX provides scalability and resilience by replicating and distributing the NIB between multiple running instances.

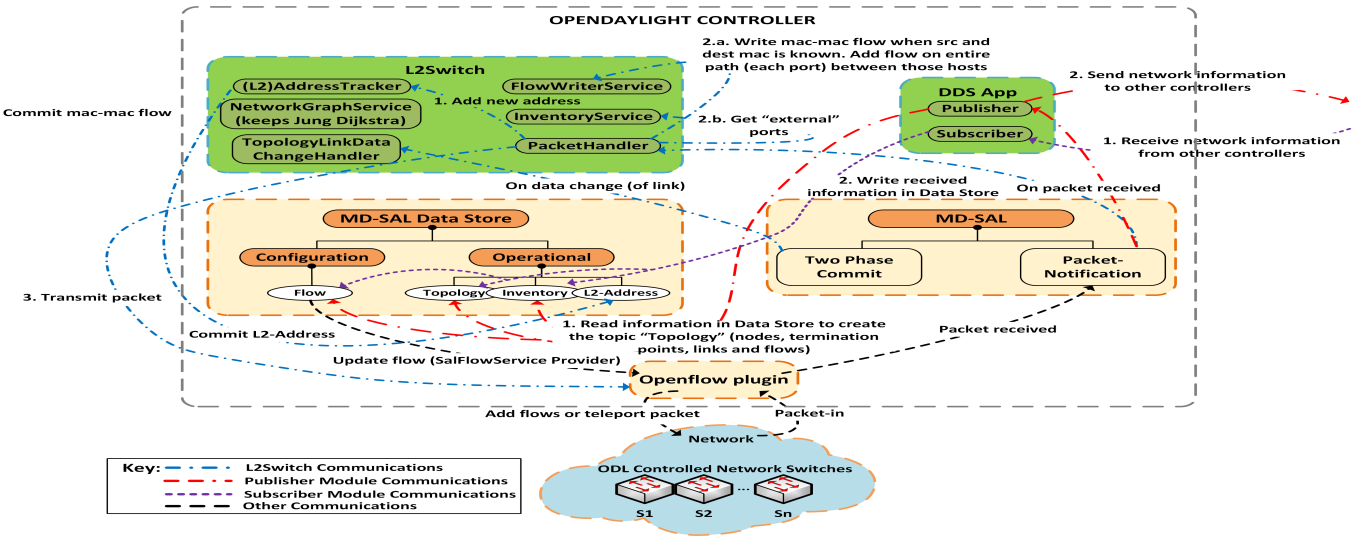



Fig. 1: Communication of L2Switch and DDS App with other modules of OpenDaylight controller.

In [4] [5] the authors propose two types of controllers: the domain and the area controllers. Furthermore, they describe the modules that compose each controller and the relationship among them. They proposed a horizontal communication module which is responsible for synchronizing global abstract network information among the domain controllers. Using a scalable NoSQL database, they store global host information, global switch information and global abstract topology information. The distribution of routing rules is realized through the Publish/Subscribe mechanism.

These approaches impose a consistent network-wide view in all the controllers and thus generate large control traffic, despite their ability to distribute SDN controllers.

There are others papers where their authors use both SDN and DDS to communicate controllers and switches. In [6] [7] [8], the authors propose DDS to exchange network state information related with QoS parameter between switches and controllers. Despite the fact that these papers use both paradigms in order to improve network behaviour, their contributions are focused on vertical communication. Besides, the authors do not take into account some characteristics of QoS in DDS to automatize the network performance.

Our purpose in this paper is to apply some parameters of QoS in the DDS to federate SDN controllers and improve their performance. Thus, in case of failure of a controller the others can assume the network control because they share the same view of the network.

## III. Integration and use of DDS App in OpenDaylight controllers

In this section, we present further details on how integrate the DDS application (DDS App) with OpenDaylight controllers. In addition, we explain some capabilities of this application in terms of QoS to improve the controller performance when occur a failure in other controllers.

In order to integrate correctly DDS App with the controller, it is necessary to install the RTI Connext DDS on the same machine where the controller will be running. After that, we generate a compatible archetype with OpenDaylight controller's application structure to our project. Once it is generated, we install the *nddsjava.jar* library and modify the *pom.xml* file to add it this dependency. This file specifies the dependencies (package of Maven and OpenDaylight repositories) that must be loaded by the application during compilation and execution times.

The main applications used in our project are L2Switch and DDS App. The first one learns about the source MAC address when a packet comes in and *teleports* the packet to the destination if it is known. Teleporting means that the packet is sent to the destination without flooding. The second one allows the communication among controllers using the Publish/Subscribe paradigm. When DDS App is installed in the controller, it creates automatically a DDS Domain, a Topic, a Publisher and a Subscriber. By means of DataWriters and DataReaders, Publishers and Subscribers can publish and subscribe DDS samples into and from the streams. More than one Topic can use the same user data type, but each Topic needs a unique name [1]. In this way, each controller would be able to communicate with one or more controllers.

Both applications use the MD-SAL Data Store to carry out their functions, it means that they do not interfere with each other. Figure 1 shows the communication of L2switch and DDS App with the MD-SAL Data Store and other modules of OpenDaylight controller.

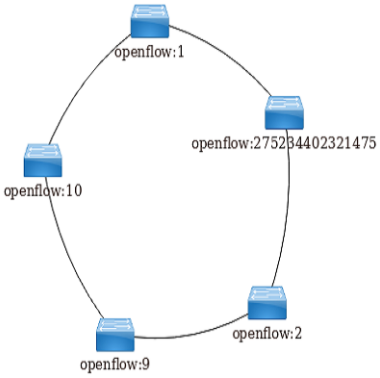
### A. Status and QoS parameters in DDS App

DDS App provides a huge number of tools to improve the network performance and the way in which the elements belonging to the same DDS Domain work.

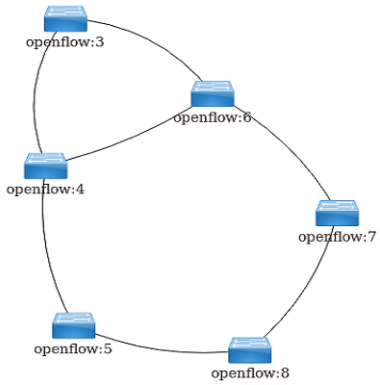
One of these tools is the *Status*. A set of status is defined for each class of Entities (Publisher, Subscriber, DataWriter, DataReader, etc.). Each type of Entities has an associated *Listener* which represents a set of functions that users may install to be called asynchronously when a determined state changes.

In our project, we focus on statuses for DataReaders, mainly **DATA_AVAILABLE Status** and **LIVELINESS_CHANGED Status**.

The first one indicates that a new data is available for the DataReader, it means that a new DDS sample has been

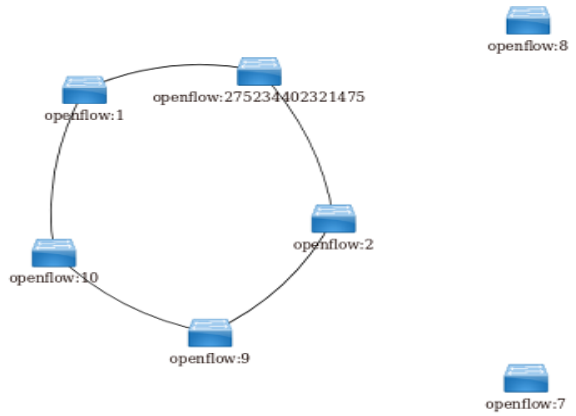



(a) Controller C0

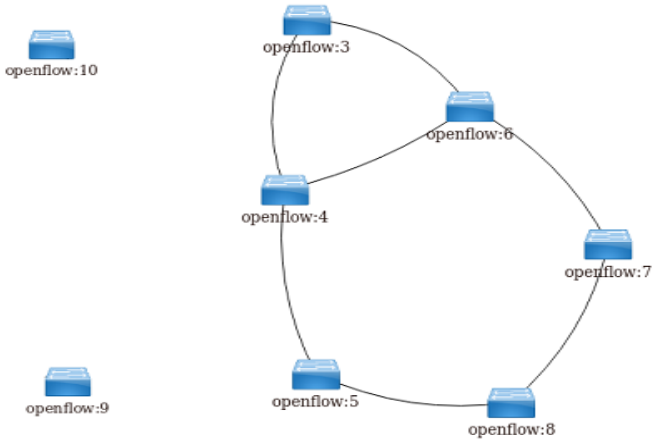



(b) Controller C1

Fig. 2: Controller's network view before being federated.




(a) Controller C0




(b) Controller C1

Fig. 3: Controller's network view after being federated.

received. The DataReaderListener's **on_data_available()** is invoked when this status changes. In this way, we avoid to poll the DataWriter what becomes in time and resource savings. This is extremely important in order to fulfill the latency requirements of 5G Networks.

The second one specifies when one or more matched DataWriters has changed their liveliness (DataWriter has become alive or not alive). The DataReaderListener's **on_liveliness_changed()** is invoked when this status changes. In this way, we will know that either a controller has failed and its nodes must be reassigned or the application has failed and it has to be restarted in order to recover the communication with other controllers. This is an important characteristic to take into account in order to improve the network response in cases of emergency in 5G Networks.

The mechanism of determining liveliness between DataWriter and DataReader is specified in the *LIVELINESS QoS* parameter. We configure how the DataWriter maintains liveliness with the DataReader and how fast the DataReader is able to detect when the DataWriter is unable to send data.

## IV. Evaluation of our design

In this section, we evaluate the performance of our design using an emulated SDN-based network. We first describe the Mininet testbed that we use to carry out the evaluation, and then present our experimental results.

### A. Mininet Test-bed

We have taken into account the 5G system architecture defined by the 3GPP in its specification TS 23.501 [9] to deploy a control plane of a 5G Network based on SDN Controllers. These controllers will be used to bridge between the control and user planes, in this case specifically, between the Session Management Function (SMF) and the User Plane Function (UPF). In our simulations each UPF will be implemented as an Openflow switch. A different group of switches have been assigned to each controller as shown in Fig. 2.

### B. Experiment Evaluations

When the controllers receive a PacketIn from their switches, they begin to exchange its network information. Figure 3 shows the network views of both controllers after being federated.

It is important to emphasize how both controllers add only the border switches to its topologies which is consequent with the way of determining the external ports in the network. Both controllers find the list of all node-to-node internal ports (List#1) by looking at all the links from the Topology data. After that, they find the list of all ports in the network (List#2) by looking at the Inventory data. At this point, both controllers have the complete network information because they obtained it through the DDS App as it is shown in Fig. 4. The difference between List#1 and List#2 is a list of external ports and switch-to-controller ports. Switch-to-controller ports have special properties and can be easily removed from List#3. This

| Node Id | Node Name | Node Connectors | Statistics |
|---|---|---|---|
| openflow:5 | | 4 | Flows \| Node Connectors |
| openflow:4 | | 5 | Flows \| Node Connectors |
| openflow:3 | | 4 | Flows \| Node Connectors |
| openflow:2 | None | 4 | Flows \| Node Connectors |
| openflow:275234402321475 | None | 4 | Flows \| Node Connectors |
| openflow:9 | None | 5 | Flows \| Node Connectors |
| openflow:8 | | 5 | Flows \| Node Connectors |
| openflow:7 | | 5 | Flows \| Node Connectors |
| openflow:6 | | 5 | Flows \| Node Connectors |
| openflow:10 | None | 5 | Flows \| Node Connectors |
| openflow:1 | None | 4 | Flows \| Node Connectors |

(a) Controller C0

| Node Id | Node Name | Node Connectors | Statistics |
|---|---|---|---|
| openflow:5 | None | 4 | Flows \| Node Connectors |
| openflow:4 | None | 5 | Flows \| Node Connectors |
| openflow:3 | None | 4 | Flows \| Node Connectors |
| openflow:275234402321475 | | 4 | Flows \| Node Connectors |
| openflow:2 | | 4 | Flows \| Node Connectors |
| openflow:9 | | 5 | Flows \| Node Connectors |
| openflow:8 | None | 5 | Flows \| Node Connectors |
| openflow:7 | None | 5 | Flows \| Node Connectors |
| openflow:6 | None | 5 | Flows \| Node Connectors |
| openflow:10 | | 5 | Flows \| Node Connectors |
| openflow:1 | | 4 | Flows \| Node Connectors |

(b) Controller C1

Fig. 4: Controller's node tables after being federated.

gives us the list of external ports.

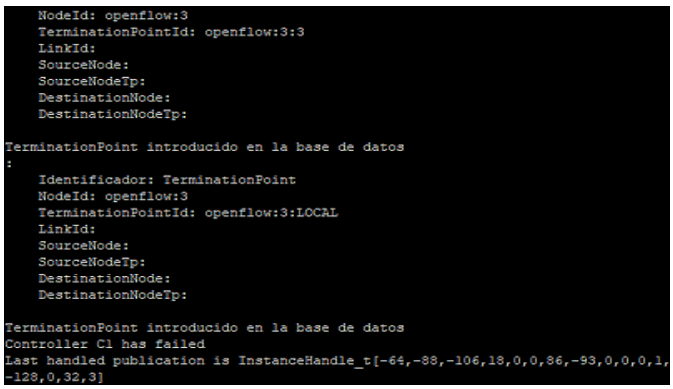
On the other hand, as part of testing the configured QoS parameters and status of DDS App, we have turned off one of the controllers to see how the other replies. The results are shown in Fig. 5. As we can see, Controller C0 recognizes that Controller C1 has failed. After that, it prints a notification and the information about the last publication sent by this controller.

## V. Conclusions

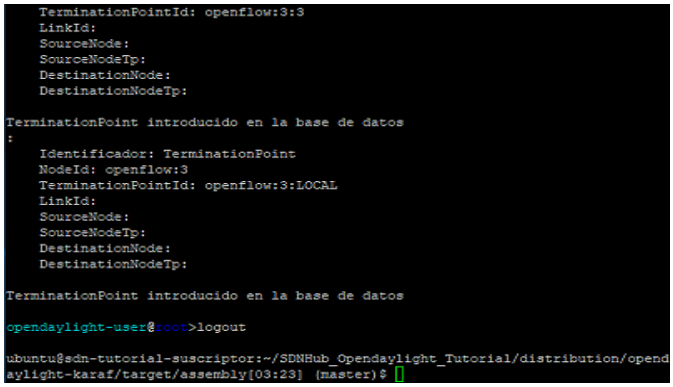
We have demonstrated how DDS App can improve the reliability and scalability of a network that is based on SDN OpenDaylight Controllers as it enables the communication between them to exchange network information. At the same time, we have implemented a mechanism to detect when a controller fails using QoS parameters and Status of DDS App. These characteristics enhance the time response of a network which is extremely important in order to use it for emergency cases and services in 5G networks.


## Acknowledgements

This work has been supported by the Ministerio de Economía y Competitividad of Spain under project TEC2016-76795-C6-1-R, TEC2016-76795-C6-4-R and AEI/FEDER, UE.


(a) Controller C0




(b) Controller C1

Fig. 5: Controller's terminal.

## References


[1] I. Real-Time Innovations, "Rti connext dds core libraries user's manual 5.2.3." http://community.rti.com/static/documentation/connext-dds/5.2.3/doc/manuals/connext_dds/html_files/RTI_ConnextDDS_CoreLibraries_UsersManual/index.htm#UsersManual/title.htm%3FTocPath%3D_____1, 2016. [Online; accessed 28-March-2018].

[2] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pp. 3–3, 2010.

[3] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks.," in *OSDI*, vol. 10, pp. 1–6, 2010.

[4] Y. Fu *et al.*, "Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks," in *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pp. 569–576, IEEE, 2014.

[5] Y. Fu *et al.*, "A hybrid hierarchical control plane for flow-based large-scale software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 117–131, 2015.

[6] L. Bertaux, A. Hakiri, S. Medjiah, P. Berthou, and S. Abdellatif, "A dds/sdn based communication system for efficient support of dynamic distributed real-time applications," in *Distributed Simulation and Real Time Applications (DS-RT), 2014 IEEE/ACM 18th International Symposium on*, pp. 77–84, IEEE, 2014.

[7] A. Hakiri and A. Gokhale, "Data-centric publish/subscribe routing middleware for realizing proactive overlay software-defined networking," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pp. 246–257, ACM, 2016.

[8] H.-Y. Choi, A. L. King, and I. Lee, "Making dds really real-time with openflow," in *Proceedings of the 13th International Conference on Embedded Software*, p. 4, ACM, 2016.

[9] 3GPP, "TS 23.501- System Architecture for the 5g System; Stage 2." http://www.3gpp.org/ftp/Specs/archive/23_series/23.501/23501-f00.zip. [Online; accessed 01-April-2018].