A DDS/SDN Based Communication System for Efficient Support of Dynamic Distributed Real-Time Applications

Lionel Bertaux, Akram Hakiri, Samir Medjiah, Pascal Berthou, Slim Abdellatif

A DDS/SDN Based Communication System for Efficient Support of Dynamic Distributed Real-Time Applications

Lionel Bertaux [a,b], Akram Hakiri [a,b], Samir Medjiah [a,c], Pascal Berthou [a,c], Slim Abdellatif [a,d]
[a] CNRS, LAAS, 7 avenue du colonel Roche, F-31400
[b] Univ de Toulouse, LAAS, F-31400
[c] Univ de Toulouse, UPS, LAAS, F-31400
[d] Univ de Toulouse, INSA, LAAS, F-31400
Toulouse, France
{lbertaux, akram.hakiri, medjiah, berthou, slim} @ laas.fr

***Abstract* — Many distributed real-time applications have dynamic requirements regarding communication delay and bandwidth. The Data Distribution Service (DDS) middleware is a key enabling technology used to support such applications. Indeed, the publish/subscribe distribution model of DDS with the ability to assign dynamic QoS (Quality of Service) parameters to DDS distribution services is able to take into account changes in the exchanged data flows and in the required QoS. This variability is taken into account at the middleware level to adjust some of DDS QoS mechanisms but is rarely propagated to the network layer to provide dynamic network communication services that fit the varying DDS distribution service needs. Usually, an over provisioned network is used, leading to network resource wastage. This paper addresses this issue and proposes a communication architecture that combines DDS with a new emerging class of communication networks named Software Defined Networks (SDN) to support efficiently dynamic distributed applications. SDN bring flexibility to the network and enable the provision of on-demand dynamic network communication services.**

Keywords— Data Distribution Service, Software Defined Networking, Openflow, Quality of Service, Distributed Simulation

I. INTRODUCTION

Distributed real-time applications are dynamic in nature, i.e. the data flows that are exchanged and their Quality of Service (QoS) requirements vary over time. Moreover, some applications operate in changing environments with heterogeneous mobile nodes involved that communicate via wireless communication links. Air traffic management, power grid control, network games, live simulation, and distributed interactive simulation fall into this category. For instance, if we consider the case of distributed interactive simulation for vehicle driver training with many networked driving simulators that evolve in a shared virtual world, each simulator needs to get some state information (position, speed, etc.) from nearby simulators. The closer the simulator, the more stringent the required delay to get information is. Movements of driving simulators (in the virtual world) introduce dynamicity in the data flows that are delivered/consumed by each simulator and on the QoS requirements related to the delivery of these data flows.

The dynamic nature of these applications is one of the major challenges for the underlying communication system and in particular for the communication network that adds other challenges such as scalability and high QoS support. Indeed, if network resource utilization is a concern, the network must be flexible enough to be reprogrammed in accordance with any change in the application. Current approaches neither address the dynamicity of applications nor care about resource utilization. They are either based on static and overprovisioned dedicated networks or overlay networks (that are difficult to set up on demand and costly), or require from the application to adapt to network's performance. The main reason is that current networks are rather static because of their complexity, the specificities of each network device, etc. An emerging class of communication networks named Software Defined Networks (SDN) have the ability to build flexible networks whose behavior can be programmed and reprogrammed on demand and in a fine grained manner. They are a promising answer to the efficient support of dynamic distributed real-time applications from a network resource utilization perspective

The main goal of this work is to propose a communication system that combines the OMG Data Distribution Service (DDS) middleware with Software Defined Networks (SDN) to efficiently support dynamic real-time applications distributed on non-dedicated network infrastructures. The rationale behind using DDS and SDN is the following. DDS has been devised (and successfully used) for high performance distributed real-time systems. It offers many QoS parameters that can be used to express the requirements on the data distribution services needed by the application. Most are dynamic and can be changed during run-time, enabling DDS to capture dynamically application QoS requirements. The Publish/Subscribe model of DDS captures the second facet of

the dynamicity of the application and allows changes in the data flows that are exchanged by the application's components.

To enforce this QoS, end-host and network resources control is mandatory. For the former, since DDS resides on end-hosts, it defines a set of mechanisms that implement QoS aware control and access to host resources. For the latter, interfaces exposed by the network are needed to manage its resources according to applications needs. This is exactly the opportunity brought by SDN.

This paper is organized as follow. In section II, DDS and SDN concepts are briefly exposed. Then, section III gives a brief state of the art concerning the concept of Application Driven Networking. Section IV details some relevant case studies. Section V explains our proposed architecture. Finally, section VI concludes this paper.

II. BACKGROUND

A. Data Distribution System (DDS)

The OMG DDS specification defines a standard architecture for data exchanges in pub/sub systems. DDS provides a global data store in which publishers and subscribers respectively write and read data. DDS provides a flexible and 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) *message flow*, by providing message-based data-centric connection management, and (5) *platform*, by supporting a platform-independent model that can be mapped to different platform-specific models, such as C++ running on VxWorks or Java running on Real-Time Linux.

The DDS architecture consists of two layers. The Data-Centric Publish Subscribe (DCPS) layer provides efficient, scalable, predictable, and resource-aware data distribution. The Data Local Reconstruction Layer (DLRL) provides an object-oriented facade atop the DCPS so that applications can access object fields rather than raw data and defines navigable associations between objects.

Figure 1 depicts the relation between DDS entities; domains, domain participants, topics, publishers, data writers, subscribers, and data readers. DCPS layer supports a global data store (DDS domain) where publishers write and subscribers read data, respectively. A DDS domain represents a virtual global data-space; information provided in the domain are accessible by the applications registered to that domain.

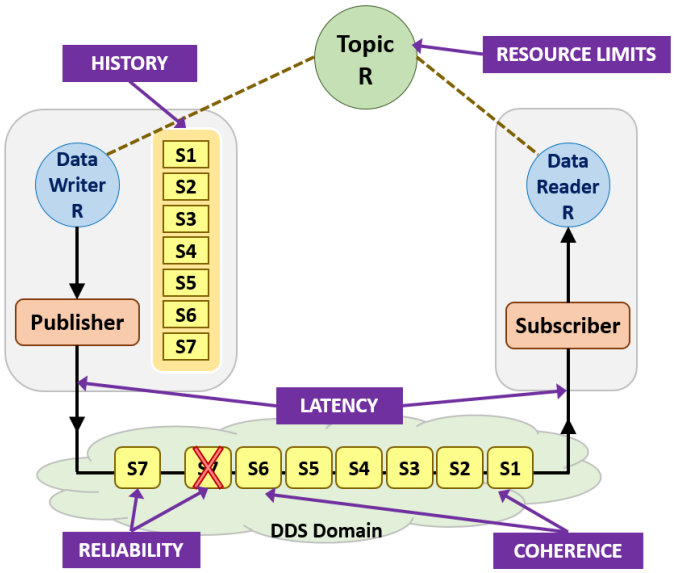

Figure 1. A View of the Data Distribution Service

DDS is topic-based, allowing strongly-typed data dissemination. A DDS *topic* describes the type and structure of the data, *data readers* and *data writers* can respectively subscribe and publish to specific topics. *Publishers* manage one or more data writers and *subscribers* manage one or more data readers. Publishers and subscribers can aggregate data from multiple data writers and readers for efficient transmission of data across a network. Publishers and subscribers discover each other automatically and match whenever they have compatible topics and QoS. DDS can use multiple topic samples called instances that are differentiated by their associated unique key. Topics can be configured with a wide range of DDS QoS capabilities imposed by peers. A DDS QoS policy can be viewed as a function that returns the state of the communication since it enables changing some aspects of the message exchange, such as connection and error handling.

Moreover, topic samples are exchanged between peers within the global data space according to a contract established in the discovery phase. The discovery process is managed by the Real-Time Publish-Subscribe (RTPS) protocol. During the discovery process each domain participant maintains a local database about all the active DataWriters and DataReaders that are in the same domain. Additionally, the discovery mechanism known as Simple Discovery Protocol (SDP) includes two phases: (1) Simple Participant Discovery which is performed by the Simple Participant Discovery Protocol (SPDP), where remote domain participants learn about each other by sending participant declaration messages (also known as participant DATA sub-messages) and (2) Endpoint Discovery which is performed by the Simple Endpoint Discovery Protocol (SEDP), in which DataWriters and DataReaders exchange information (i.e., such as QoS, data types, etc.) to match each other by sending publication/subscription declarations in DATA messages that we will refer to as *publication DATAs* and *subscription DATAs*.

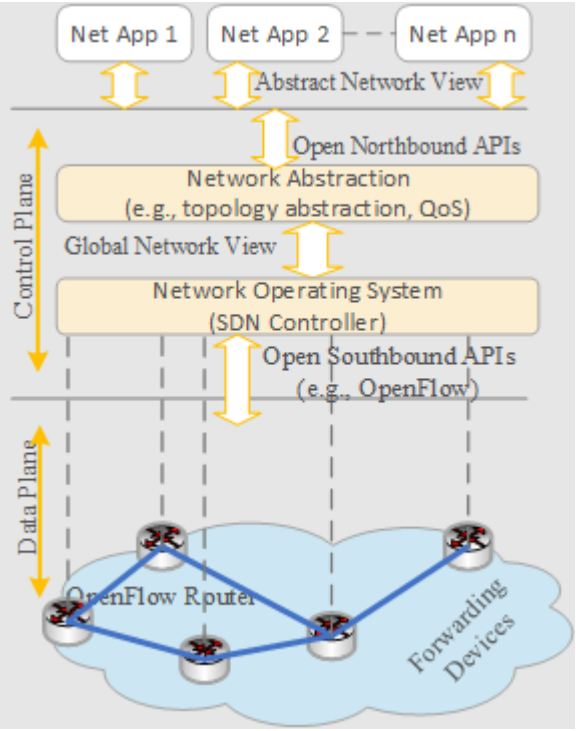

Figure 2. Overview of the software-defined networking (SDN)

B. SDN concepts and OpenFlow

Recently, Software-Defined Networking (SDN) [9] has emerged as a new approach for network programmability and management, where the centralized control plane logic is decoupled from the forwarding plane as shown in Figure 2. Network programmability is a software approach to dynamically control, manage, configure or even use a network. Amongst others, it covers the existence of open interfaces for device configuration, the capability to design virtual overlay networks or the opportunity to easily deploy network functionalities. In an SDN, centralized control uses the network-wide view of the network to ease management and configuration of data-path for network services such as resource allocation or virtual network deployment. Such networking concept can help migrate legacy networks to vendor-free platform adaptable to each user needs that could be seen as a Network as a Service (NaaS).

In Software-Defined Networks (SDNs), control plane is logically separated from data plane. SDN architectures define a new entity that centralizes control intelligence of one or more network elements (basically switches). A protocol is then defined to communicate between control plane and data plane. Examples of such protocols are OpenFlow [3] or ForCES [8].

In an OpenFlow-based SDN architecture, the control entity is called a controller. The controller offers northbound interfaces to network applications and southbound interfaces to communicate with data plane. OpenFlow [3] is one of the possible southbound protocols. OpenFlow behavior is simple but it can allow complex configurations: the hardware processing pipeline from legacy switches is replaced by a software pipeline based on flow tables. These flow tables are composed by simple rules to process packets, forward them to another table and finally send them to an output queue or port. Flow tables can be built in a proactive or in a reactive way.

With such functionalities, OpenFlow can be used to implement Quality of Service policies in a switch by dynamically managing flow differentiation. For example, a DiffServ-like architecture can be easily deployed in a switch with several queues per output port and differentiation can be applied based on various criteria (destination address, sender address, port numbers, etc.). An important point is that OpenFlow is not in charge of configuring queues attached to output ports of a switch. To this end, OpenFlow Configuration (OF-Config) protocol [4] has been defined based on NETCONF and manages the "OpenFlow context": establishing links between switches and controller, configure output port in switches, etc. Since version 1.3 and the introduction of a meter table, OpenFlow is able to monitor flow rates and perform actions if a predefined limit is exceeded: drop packets or mark the DSCP field. The drop policy is efficient but very aggressive and can lead to the introduction of data bursts for TCP flows [1]. Globally, OpenFlow QoS capabilities are considered incomplete to implement an effective QoS architecture and the protocol needs some external features to be fully functional [2] (e.g. configuration of outgoing queues in switches).

However, full network configuration is possible by dynamically installing flow processing rules in switches with OpenFlow and configuring outgoing queues with another application. These kind of configurations can be managed by an application on the northbound interface of a controller, the latter being performed with OF-Config or a legacy protocol like SNMP. Another approach to implement QoS with OpenFlow is to perform flow splitting and to assign specific routes to given flows. Coupled with an optimization algorithm, such mechanisms can improve the quality of the communication in the case of video streaming [5]. Virtual networks (virtual switches and virtual links) can also be deployed by "slicing" the OpenFlow network. An intermediate controller acts as a "proxy" for the real control plane located above him and provides each controller with a virtual network (e.g. FlowVisor [6] and more recently OpenVirteX [7]).

III. Related Work

Prior middleware solutions for network QoS management focus on how to add network QoS services for CORBA-based communication [10]. A large-scale event notification infrastructure for topic-based pub/sub applications has been suggested for peer-to-peer routing overlaid on the Internet [11]. Those approaches can be deployed only in a single-domain network, where one administrative domain manages the whole network. Extending network QoS solutions to the Internet can result in traffic specified at each end-system being

dropped by the transit network infrastructure of other domains [12]. For example, authors in [13] have presented a network communication broker to enable per-class QoS for multimedia collaborative applications. Even if this network broker enhanced the QoS allocation by differentiating the traffic processing at the network edges, it supports neither mobility service management nor scalability since it adds complicated interfaces to both applications and middleware for the QoS notification.

Our prior work [14] on Velox used an MPLS tunneling mechanism to propagate DDS QoS over an overlay model. Velox statically created a logical tunnel between remote DDS participants to conduct QoS negotiation and resource reservations. Any time communication failed, the user had to reconfigure the network manually. Moreover, application requirements could not change during runtime communication between participants. To overcome these limitations, we introduced the NetQSIP framework [15] to enhance prior research on QoS management at the network layer by integrating QoS along two key dimensions: (1) the horizontal direction, i.e., between different adjacent layers in the application, middleware, and network, and (2) the vertical direction, i.e., within a particular layer. In particular, our NetQSIP framework maps application flow requirements into the DDS layer to allow end-to-end QoS provisioning. Hence, NetQSIP provides dynamic QoS management and simplifies resource allocation by using SIP to automate end-to-end QoS provisioning.

However, both Velox and NetQSIP were integrated on legacy network devices so their deployment depends on the internal capabilities of each network fabric, which the internals differ from vendor to vendor, with no open software platform to experiment new ideas. These devices have their control and forwarding logic parts both integrated in monolithic, closed, and mainframe-like boxes. Consequently, only a small amount of external interfaces are standardized (e.g., packet forwarding) but all of their internal flexibility is hidden.

Likewise, the work presented by [16] outlines an extension to the IETF REsource Location And Discovery (RELOAD) protocol [17] for content discovery and DDS messages transfer over large-scale legacy networks. Although this paper addressed the discovery scalability issue of DDS, it centers on a structured P2P overlay architecture in legacy WANs, which is different from our work described in this paper. The work described in this paper is based on SDN programmable networks, where DDS is used atop of the SDN northbound interface to control network devices and program QoS in flexible and scalable way.

IV. CASE STUDY

This section intends to give an example of what could be the benefits of using Software Defined Networking for Distributed Real-time applications. Requirements for this class of applications are stringent and some of them will be explored when using SDN concepts.

However, real-time communication could not be provided without dedicated network architecture. To make clear the discussion, we propose to take an example of a multi-site company network given in Figure 3. A Virtual Private Network between multiple offices of the same company is considered: the headquarters (A) are connected to the other offices (B) and (C) with a Virtual Ethernet Service. Then, business applications and services are running as if they were deployed over a classical LAN.

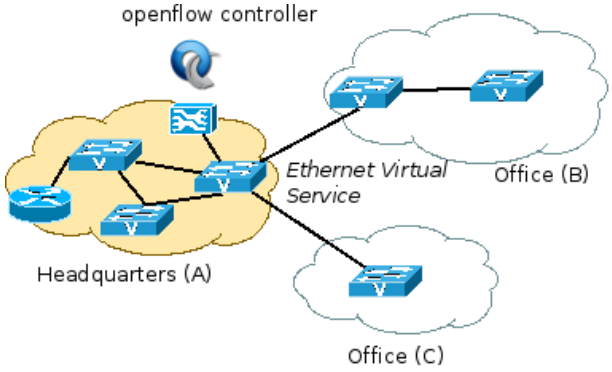

Figure 3. Multi-site network architecture

The network is then essentially composed of switches, few routers, and middle-box appliances.

This kind of network architecture is simple and easy to maintain. Links between sites have to be provisioned according to the distributed applications needs, but also to the generic communication (telephony, network appliances, etc.).

However, the lack of traffic isolation and the difficulty to deploy complex quality of service policies in the network are the counter part of the simplicity. It is then not possible to ensure the communication requirements of real-time applications and especially if requirements are stringent and variable. Overprovisioning is then the only solution, but at a high cost.

A. Software Defined Networking benefits

Using SDN concepts could knock down the technological barriers that slowdown the dissemination of real-time distributed applications. The main idea in this paper is to combine objects and communication descriptions provided when using the DDS middleware, especially related to Quality of Service, and SDN concepts as a flexible solution to configure the network with dedicated and flexible QoS solutions.

The benefits are multiple and this article proposes to explore them following three use cases.

First, generally speaking and without any consideration on using a middleware, the management of a distributed network across several entities is complex. Providing an acceptable level of service to distributed real-time applications, but also offering enough bandwidth to network appliance is a complex tradeoff that have to be implemented with a fine-tuned QoS policy. The problem is hardened when considering the evolution of the requirements over time: the policy has to be

adapted frequently and consistently. It mainly consists in collecting the communication requirements from the applications, defining the new policy, and scripting the control of network switches with SNMP for instance. This could be a hard task, as the consistency of the configuration must be ensured on all devices and at the same time. With a unique control point (the controller) OpenFlow provides a robust solution to this task. Application requirements could be collected within the controller thanks to the centralized description of the application (DDS configuration file), an optimal policy could be computed and deployed thanks to the OpenFlow protocol.

The next use case, extends the previous one related to the configuration, and addresses the automatic network slicing. When several applications are coexisting within the same network, it is sometime relevant to segregate traffics from each application. Several VPN are then configured to ensure that bandwidth resources from one application would not be consumed by another application. OpenFlow provides solutions for network isolation, called slicing that simplifies this operation. The switches could monitor network resource consumption and forwarding decisions could be taken on the go by the controller.

Finally, the last use case takes benefit from DDS smart capabilities for data diffusion. Commonly, broadcasting is massively used over a LAN to maintain the consistence of the distributed application. However, considering a virtual Ethernet service over private leased line it could be inefficient and costly when resources are scarce. DDS offers a filtering mechanism that only propagates data to the concerned subscribers instead of broadcasting. However, this mechanism is useless over a classical Ethernet network as there is no way to filter data. It is possible to implement a dynamic filtering mechanism with OpenFlow that reduce the traffic between interconnected switches at its minimal value.

B. Discussion

The combination of DDS middleware capabilities with the OpenFlow SDN concepts is promising. However, this is not possible without an appropriate architecture that defines the role and the place of the different entities. The next section is devoted to the architecture definition.

V. PROPOSED ARCHITECTURE

This section presents the details of the network architecture required to deploy enhanced QoS mechanisms for DDS over SDN-enabled networks.

A. Targeted QoS parameters

The network quality of service can be described through multiple parameters related to different aspects of the data transmission such as:

- *Transmission errors* (bit error rate, packet error rate, packet loss, FEC level, etc.)
- *Transmission delay* (end-to-end delay, jitter, queuing delay, first packet arrival, etc.)
- *Transmission throughput* (bandwidth, packet success, etc.)

Through DDS, applications can specify their QoS requirements by the mean of various QoS parameters. Indeed, DDS allows the specification of the following QoS policies:

Table 1. QoS Parameters in DDS

Volatility	– *Durability* – *History* – *Reader data lifecycle* – *Writer data lifecycle* – *Lifespan*
Infrastructure	– *Entity factory* – *Resource limits*
Delivery	– *Reliability* – *Time based filter* – *Deadline* – *Content filters*
User QoS	– *User data* – *Topic data* – *Group data*
Presentation	– *Partition* – *Presentation* – *Destination order*
Redundancy	– *Ownership* – *Ownership strength* – *Liveliness*
Transport	– *Latency budget* – *Transport priority*

In the DDS middleware, these QoS parameters are handled in different ways and at different layers of the communication stack. For example the "History" parameter related to "Volatility" is handled at the highest layer (sub-application layer). Whereas, other parameters such as "Latency" or "Transport priority" can be handled at the network layer or below.

On the other side, QoS management is implemented within OpenFlow. QoS management is handled via 3 techniques: (a) per-port queue management, (b) per-flow priority, and (c) queue and flow statistics. The QoS management in OpenFlow has been progressively introduced through the consecutive versions. OpenFlow v1.0 brought the specification of the minimum rate for a port queue, while OpenFlow v1.2 brought the specification of the maximum rate for a port queue and the specification of priorities among flows. Finally, OpenFlow v1.3 brought Meter tables for statistics purposes.

In order to enhance DDS applications running on top of SDN-enabled networks, it is important to translate all the DDS QoS parameters into L2/L3 policies that can be implemented in the network.

B. QoS parameters capture

Before translating the DDS QoS parameters into L2/L3 policies, it is important to capture all the relevant parameters in the most seamless way for the running applications. To this end, we can envision two different approaches, both requiring no modification in high level application:

(1) Middleware-independent QoS parameters capture:
QoS parameters can be captured using the DDS monitoring service. With the monitoring service, the SDN controller subscribes to a discovery DDS topic and thus, takes part of the DDS domain. It then notifies QoS parameters of any running DDS-based application. This approach is transparent to the application. However, it requires the controller to be part of the DDS domain and thus, it must run the DDS middleware.
A more seamless way would be to capture the QoS parameters specification by monitoring the traffic among the DDS entities within the network. This is very challenging and may be less efficient since it involves the reverse engineering of the DDS protocols (signaling, data, etc.).

(2) Middleware-dependent QoS parameters capture:
In this approach, the DDS middleware should be augmented with a new SDN module. As illustrated in Figure 4, this module will communicates with the Network Application which is running on top of the SDN controller in order to provide it with all the relevant QoS information to optimize and/or reconfigure the underlying L2/L3 network topology.

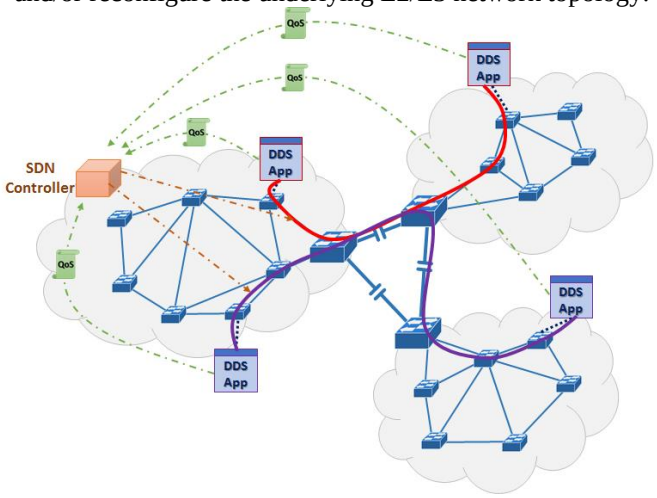

Figure 4. DDS QoS parameters capture to L2/L3 flows orchestration

C. L2/L3 flows orchestration

In a SDN, L2/L3 policies can be applied by passing high level rules to the controller that can distribute and apply them to data plane. Available functionalities and configurations depend on the protocol used to communicate with switches and configure them. With OpenFlow, a controller is able to program the processing pipeline in switches by populating successive flow tables. Several functionalities can thus be deployed by performing specific L2/L3 treatment:

- Basic L2/L3 routing:
  - L2/L3 header treatment,
  - Queuing in adequate outgoing queue/port.
- Flow splitting:
  - Path selection depending on flow,
  - Possible dynamic load balancing.
- Multicast with RP-point:
  - Data-duplication in defined switches,
  - Multiple points of duplication.
- Adaptive sending rates:
  - Limit rate on specific flows/ports.
  - Drop / Redirect / Alert if overtaking rate.

Furthermore, a network managed via OpenFlow can be "sliced" to create virtual networks. It is thus possible to deploy virtual switches and links with a topology adapted to current state and active DDS flows: point-to-point, mesh, tree, etc.

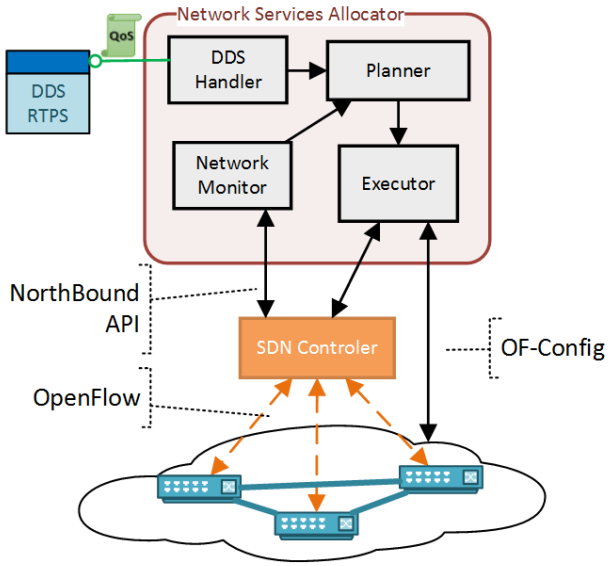

Figure 5. Network application in charge of configuration.

D. Network ressources allocation

In our design, we chose to build an application located on top of the controller NorthBound interface, since it is more compliant with SDN standards than deploying an extension to a controller (see Figure 5). Globally, this application is responsible for translating DDS QoS constraints into forwarding rules that can be implemented in network switches. It is composed by 4 logical blocks: a DDS information handler, a network monitor, a planner and a command executor. Interactions and behavior of theses logical blocks are summarized in Figure 6. The DDS handler is connected to DDS domain and uses RTPS to collect data on DDS flows (required bandwidth, delay tolerance, data generation model ...). The network monitor element uses the SDN controller NorthBound API to gather information on the underlying network (topology, capacity ...) and measurements on active flows (rate, counter ...). Based on monitoring data from previous elements, the planner computes the optimal set of network resources that is needed to support DDS flows with their required QoS. Optimization may induce splitting point-to-multi-point flows at RP-points (Rendezvous point), adapting DDS publishing rates to subscriber needs by shaping network flow rate or load balancing flows over different paths while preserving delay requirements. The last element is in charge of applying the network configuration built by the

planner in the real world either via the controller NB-API for OpenFlow related rules or via OF-Config for switch configuration (Ex: configuring queues attached to outgoing ports).

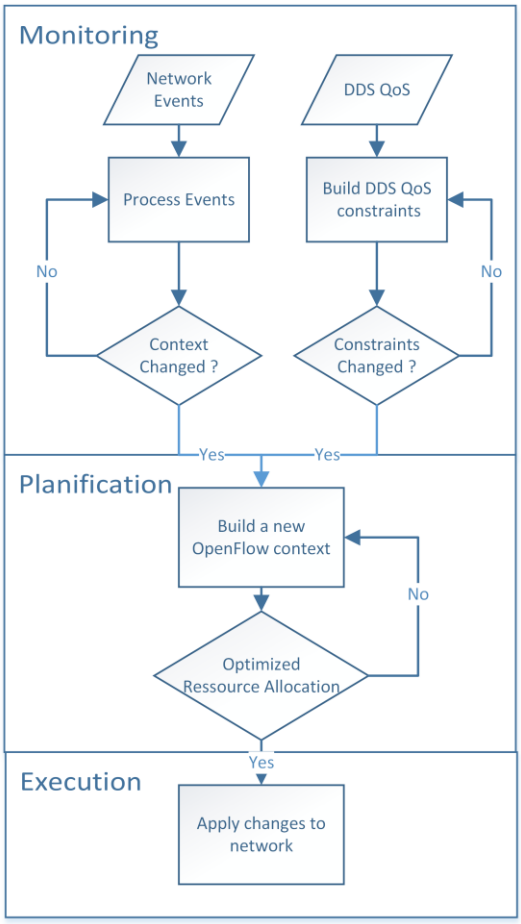

Figure 6. Network services allocation algorithm.

In order to check the technical feasibility of our solution, we considered of the OpenFlow controller Floodlight [18]. Based on the REST communication architecture, the NB-API of Floodlight allows an application to capture information from the network and to set up various rules in the data plane (see Table 2). These requests are specific to the use of OpenFlow and take into account the possibilities of this South Bound protocol. They can help designing high level functionalities needed by our proposition:

- Build network topology based on known switches and links with accurate information such as port speed (i.e. path bandwidth),
- Monitor network events by periodically checking available links and switches,
- Monitor network flows by periodically checking traffic counters,
- Reserve resources on a path between subscriber and publisher by adding a static flow,
- Establish a dedicated network for data flows of specific topics by adding a virtual network.

These functionalities are executed by the Network Monitor and the Executor modules of our Network Services Allocator (Figure 5) to achieve their respective goals: obtain status and statistics from the network and control the data-forwarding elements of the network.

Table 2. Examples from the Floodlight Northbound-API

Description	REST Method	Possible Options
List traffic counters	GET	Per controller / per switch
List all switches	GET	
List all links	GET	Internal / External domain
Add a static flow	POST	Arguments and output
Add virtual network	PUT	ID and possible Gateway

VI. Conclusion

In the near future, software defined components will be a predominant part in communication networks thanks to the growing use of virtualization (resources or network). The design presented in this paper shows how SDN computer networks can be used to provide dynamic customized network services to dynamic DDS applications with efficient network resource utilization. The feasibility of this solution has been demonstrated through the description of its components along with their functionalities and how they can be implemented.

Acknowledgements

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] Mohan, P.M.; Divakaran, D.M.; Gurusamy, M., "Performance study of TCP flows with QoS-supported OpenFlow in data center networks," Networks (ICON), 2013 19th IEEE International Conference on , vol., no., pp.1,6, 11-13 Dec. 2013

[2] Sonkoly, B.; Gulyas, A.; Nemeth, F.; Czentye, J.; Kurucz, K.; Novak, B.; Vaszkun, G., "On QoS Support to Ofelia and OpenFlow," Software Defined Networking (EWSDN), 2012 European Workshop on , vol., no., pp.109,113, 25-26 Oct. 2012

[3] Open Networking Foundation, OpenFlow Switch Specification – version 1.4.0 (Wire Protocol 0x05), October 2013.

[4] Open Networking Foundation, OF-CONFIG 1.2 – OpenFlow Management and Configuration Protocol, 2014.

[5] Egilmez, H.E.; Civanlar, S.; Tekalp, A.M., "An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks," Multimedia, IEEE Transactions on , vol.15, no.3, pp.710,715, April 2013

[6] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. 2010. Can the production network be the testbed?. In Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI'10). USENIX Association, Berkeley, CA, USA, 1-6.

[7] Ali Al-Shabibi, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, William Snow, Guru Parulkar, "OpenVirteX: A Network Hypervisor," in Open Networking Summit, March 2014.

[8] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern. Forwarding and Control Element Separation (ForCES) Protocol Specification. RFC 5810 (Proposed Standard), March 2010.

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

[10] Dasarathy, Balakrishnan, Gadgil, Shrirang, Vaidyanathan, Ravi, et al. Adaptive network QoS in layer-3/layer-2 networks as a middleware service for mission-critical applications. Journal of Systems and Software, 2007, vol. 80, no 7, p. 972-983.

[11] Rowtron, Antony, Kermarrec, Anne-Marie, Castro, Miguel, et al. SCRIBE: The design of a large-scale event notification infrastructure. In: Networked group communication. Springer Berlin Heidelberg, 2001. p. 30-43.

[12] Balasubramanian, Jaiganesh, Tambe, Sumant, Dasarathy, Balakrishnan, et al. Netqope: A model-driven network qos provisioning engine for distributed real-time and embedded systems. In: Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE. IEEE, 2008. p. 113-122.

[13] Zhang, C and Sadjadi and S. Masoud and Sun Weixiang and Rangaswami Raju and Deng Yi, A user-centric network communication broker for multimedia collaborative computing, IEEE, CollaborateCom, 2006

[14] Hakiri, Akram, Berthou, Pascal, Gokhale, Aniruddha, et al. 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, 2013, vol. 86, no 10, p. 2574-2593.

[15] Akram Hakiri, Pascal Berthou, Aniruddha Gokhale, Douglas Schmidt and Thierry Gayraud, "Supporting SIP-based End-to-End Data Distribution Service QoS in WANs". Journal of Systems Software, 2014.

[16] Jose M. Lopez-Vega and Gonzalo Camarillo and Javier Povedano-Molina and Juan M. Lopez-Soler, RELOAD extension for data discovery and transfer in data-centric publish-subscribe environments, Journal of Computer Standars and Interfaces, Vol (36), num (1), pp 110 - 121, 2013

[17] C. Jennings and B. Lowekamp and E. Rescorla and S. Baset and H. Schulzrinne, REsource LOcation And Discovery (RELOAD) Base Protocol, IETF, RFC 6940,jan, 2014

[18] Floodlight Controller, http://docs.projectfloodlight.org/display/floodlightcontroller/The+Controller, July 2014.