Management of industrial communications slices: Towards the Application Driven Networking concept

Slim Abdellatif, Pascal Berthou, Thierry Villemur, Armel Francklin Simo Tegueu

Management of Industrial Communications Slices: Towards the Application Driven Networking Concept

Slim Abdellatif[1], Pascal Berthou[1], Thierry Villemur[1] and Francklin Simo[2]

[1] LAAS-CNRS, Université de Toulouse, CNRS, INSA, UPS, UT2J, Toulouse, France
[2] Viveris Technologies, F-31400 Toulouse, France

Abstract—To address the performance problems that many business critical applications are experiencing, network vendors and Network Service Providers (NSP) are reconsidering the integration of some form of application awareness in the way their networks forward user traffic. Their ultimate goal is to devise new network service models that are dedicated and customized to applications while efficiently using network resources. Even if this approach is not new and has been already followed with a mitigated success two decades ago, the emergence of software-Defined networking and network virtualization coupled with DPI (Deep Packet Inspection) pave the way for the investigation of new approaches/solutions towards application aware/driven networks. This is exactly the motivations of this work whose objective is to propose an Application Driven Network (ADN) that provides services to a specific type of applications, i.e. Dynamic Data Distribution Service (DDS) based applications. Considered as one of the leading connectivity standard for industrial IoT communications, focusing on DDS allows a fine-grained description of applications' traffic and needs. With this information as input, the proposed ADN is able to provision network services that stick to application needs while using network resources efficiently. This paper sketches the general architecture of such ADN by describing its main components, their requirements as well as their algorithms. This solution has been implemented, prototyped and applied to a DDS based distributed interactive simulation application.

Keywords—Application Driven Networking; Software Defined networking; Data Distribution Service; Network Virtualization

I. INTRODUCTION

Technologies like the Industrial Internet of Things (IIoT), artificial intelligence, and cloud applications are the pillars of the industry 4.0 [1]. In industries where the enterprise typically relies on heavy machinery or other physical equipments the ability to connect these assets and gather data from them can lead to benefits like improved efficiency and predictive maintenance. However, the networks could become the Achilles' heel of this new paradigm.

Despite all the advances on network Quality of Service (QoS) provisioning that we witnessed during the last decades, the performance of business critical applications on an enterprise network is still an issue that concerns an increasing number of organizations [2]. It reveals that most organizations do not have any precise knowledge on the QoS requirements of most of their critical applications. Indeed, some QoS parameters are difficult to specify and quantify, especially for applications that require strong dynamicity (i.e. data flows exchanged between application processes and their associated QoS requirements strongly vary over time). These dynamic changes are one of the main reasons of the precise QoS definition difficulty [2]. The industry is not exempt from this problem due to the emergence of new complex services. A second reason is that organizations and running applications have no precise knowledge of the network performance. Therefore, starting from user complaints and feedbacks, the QoS planning adjustments made by organizations is based on network resource over-provisioning. This strategy wastes network resources: network bandwidth requirements on enterprise networks are ever increasing at the cost of poorer average network resource utilization.

Figure 1 could be a typical example of Industry 4.0 network infrastructure [1]. Factory A & B are interconnected with a cloud hosting business applications through a public transport network. A virtual network composed of network links is established based on a Service Level Agreement (SLA) negotiated with the operator. Each link must then be provisioned according to the resources required by the application. How many resources should be affected to each link to ensure the SLA and then the application performance? Is it optimal? Will it fit with the application evolving requirements? Could the reservation be automatized? These are the issues that must be resolved by the IT community to build intelligent, dynamic, high-performance networks.

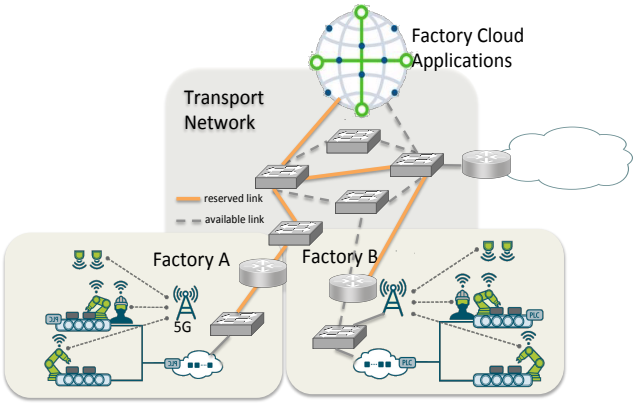

Figure 1 - Industry 4.0 networks

One way to address these problems is to introduce some form of application awareness into the forwarding behavior of computer networks. The network is able to provide customized data paths to applications, i.e. data paths that are assigned on a

per application basis (and not on a destination basis) with the required characteristics (in terms of assigned network resources) to meet application QoS needs. The network must cope with changing needs. It derives the current application needs, then computes and installs on the fly the required network resources along the data path(s). The newly created Application Driven/Aware Network (ADN) enables the provision of network services with guaranteed dynamic QoS. It reinforces the network utilization efficiency.

In fact, the association of the application requirements to the network resource utilization is not new and has already been addressed in the dense literature related to QoS provisioning and policy based networking [3]. All these works were applied to (and hence constrained by) existing computer networks where forwarding is based on destination (potentially, with a complementary priority tag). The advent of Software-Defined Networking (SDN) paves the way for new approaches towards application driven networks. Indeed, the flexible flow-based forwarding brought by the SDN paradigm allows unprecedented control on network forwarding behavior. On the other hand, Deep Packet Inspection (DPI) can be used to identify the applications and infer their short-term requirements.

There have been some recent proposals on leveraging SDN, and potentially DPI, to provide some form of application driven networking that aim at providing customized services to applications [3][4][5][6][7]. This paper proposes an ADN architecture, which in contrast to previous works, relies on a fine-grained description of applications' traffic and needs. These latter are obtained from the Data Distribution Service (DDS) middleware on top of which the applications that we are considering are built. With this architecture, it is possible to provide services that exactly or closely match application requirements. Coarse or overestimated needs can be avoided, and very efficient network resources utilization can be achieved.

This paper is organized as follows. Section II motivates and positions our work. Section III introduces the necessary prerequisites. Section IV relates a selection of domain work. Section V describes the design principles of our proposal and positions it with respect to existing works. Next, we present the general architecture of our proposed ADN as well as some of its main algorithms. Section VIII details the ADN prototype that we implemented; it also describes the application of our ADN approach to a distributed interactive simulation application. Section IX presents the performance evaluation of our main component. Finally, section X concludes the paper.

II. MOTIVATION, PROBLEMATICS AND POSITION OF OUR WORK

As said in the introduction, despite all the advances on network Quality of Service (QoS) provisioning, the performance of business critical applications running on an enterprise network is still an issue that concerns an increasing number of organizations [2]. This is even more true when talking about industrial automation domain and the concept of Industry 4.0 that relies on Internet technologies to create a smart production system. Beyond the traditional supervisory control and data acquisition system (SCADA), industrial communication solutions are heterogeneous and address new challenges. They include fieldbuses, industrial Ethernet, industrial Wireless, and local manufacturing clouds. In their 5G use cases requirements the GSM Association has identified five different application areas for Industry 4.0 vertical market [9]. The 3GPP technical specification [10] identifies more than forty uses cases in these areas and provide for all of them the intended service flow requirements. With the emergence of the video transmission, augmented reality, clouds, or massive IoT, requirements are so complex they cannot be planned as they were using cyclic data transmissions. Both information technology (IT) and telecom networks could not cope with the automation-specific needs for deterministic (hard deadlines boundaries and isochronous communications), low jitter, reliable, available, and of course low costs [8]. This is especially challenging when new application interactions and application flexibility are introduced.

These days, a generic communication model that consists of three layers of networks, middleware and application is commonly implemented. Among them, the Data Distribution Service (DDS) middleware [12] is particularly characterized with a wide-ranging set of QoS parameters that makes it one of the references for the industrial communication domain. Applications running on top of DDS can dynamically express their QoS and communication requirements through a specific API. The middleware ensures data availability, guarantees real-time response, manages complex dataflows and makes deployment flexible as long as the network infrastructure can support it. This is the main weakness of the current approaches as the current deployed networks are static and must be overprovisioned. The application has to adapt to the networking performances that cannot be changed in time.

Things are changing as the 5G architecture is expected to accommodate a wide range of use cases with advanced requirements in terms of latency, bandwidth and resilience. The concept of slice builds logical network structure on top of a same physical infrastructure to fulfill vertical specific requirements. This requires highly manageable infrastructure components to be deployed and changed on demand at runtime according to the applications needs. Thanks to the association of Software Defined Network paradigm (SDN), where one of the goals is to supply network flexibility, Monitor-Analyze-Plan-Execute (MAPE) framework, that supplies autonomic functionalities, and the power of a middleware like DDS, this work proposes an architecture that will satisfy the constraints of industrial communications while optimizing network resources. Starting from the detailed description of the application dynamics, the networking service will dynamically adapt at best the networking resources with the application requirements. This architecture should be considered as a key enabler for the deployment of future industrial communication slices. In the 5G framework, the slice implementation into the Radio Access Network (RAN) is considered out of the scope. They consider "RAN dependent" [11] the resources allocation/segregation into the RAN and it is up to the operator to fix the way to implement it. The same rule applies to local networks where our architecture can be considered as a solution to implement the slicing concept. It allows fine grains services definition that allows the network to fulfill the application requirements while optimizing the slice resources. However, even if we take into account the concept of slice, we consider beyond this work the definition of an interface compliant with the 5G framework. Even if it remains possible.

III. SOME TECHNOLOGICAL PREREQUISITES

A. Data Distribution Service (DDS)

The Object Management Group (OMG) has specified the DDS middleware to address the communication needs of real-time and high performances distributed applications. DDS implements the publish/subscribe paradigm. It is based on the abstraction of a strongly typed Global Data Space (GDS, denoted DDS domain in Figure 2) where applications push their typed data into the GDS with "DataWriters" and interested applications subscribe to these data and read them from the GDS with "DataReaders". Data types are explicitly named and defined by the DDS topics. As explained in the following example derived from the DDS tutorial, a temperature sensor could be the DataWriter of a Topic defined by a TempSensorType. Making a parallel with classes in object-oriented programming languages, a Topic can be regarded as defining a class whose instances are created for each unique value of the topic.

```
enum TemperatureScale {CELSIUS,KELVIN,FAHRENHEIT};

struct TempSensorType {
  short id;
  float temp;
  float hum;
  TemperatureScale scale;
};

dds::topic::Topic<tutorial::TempSensorType>
  topic(dp, "TTempSensor");
```

In DDS, data publications and subscriptions are decoupled in time. Publication intent, subscription and de-subscription take place at any time in the application lifetime. If a subscription matches a publication (concerns the same topic), the DDS middleware is in charge of distributing the data to the newly subscribed application. It relies on RTPS (Real-Time Publish/Subscribe Protocol) as a generic and standardized data transport protocol.

One of the strength of DDS is to provide applications with a comprehensive set of QoS parameters (named QoS policies in DDS) to express the performance (or QoS) requirements on the distribution of the data that they produce or consume. These requirements apply to the distribution of each type of data (topic). They cover reliability aspects as well as the data production or consumption rate, the transfer delay, etc. Furthermore, some of them are modifiable on the fly. Based on this principle, DPI is not required because all service flows descriptions are provided by the application to the middleware.

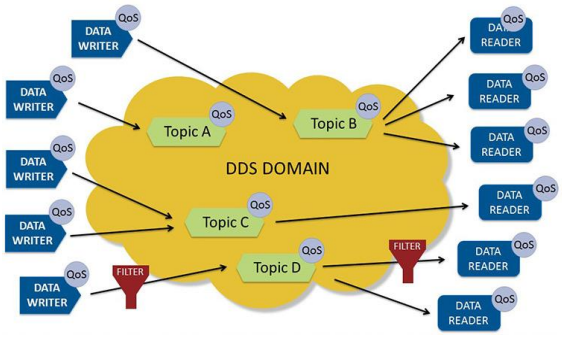

Figure 2 - Data Distribution Service overview

For each topic, the resource producer and consumers have to specify a set of QoS Policies (selected in Table 1) that the DDS middleware will manage and monitor. The QoS Policies are divided in 5 categories related mainly to the data availability during the application, the data delivery across the DDS domain and the real-time requirements. For instance, a real-time topic could be set with a volatile durability, a fixed deadline of several milliseconds and a latency budget that specifies the maximum acceptable delay from the time the data is written until the data is inserted in the receiver's application-cache. The producer and the consumer will be informed by the middleware in case of a temporal violation.

QoS Policy	Policies Type
DURABILITY	**Data Availability**
DURABILITY SERVICE	
LIFESPAN	
HISTORY	
PRESENTATION	**Data Delivery**
RELIABILITY	
PARTITION	
DESTINATION ORDER	
OWNERSHIP	
OWNERSHIP STRENGTH	
DEADLINE	**Data Timeliness**
LATENCY BUDGET	
TRANSPORT PRIORITY	
TIME BASED FILTER	**Resources**
RESOURCE LIMITS	
USER_DATA	**Configuration**
TOPIC_DATA	
GROUP_DATA	

Table 1 – DDS QoS Policies

DDS middleware addresses the industrial communications domain according to minimal configurations. Cyclic variables and events can be spread to unicast, multicast or broadcast domains. [13] has proposed a mapping between different factory automation traffic types into DDS entities (Table 2). A platform interconnecting Profibus and industrial Ethernet has been prototyped. [14] also addresses the relevance of DDS for massive communications for Industrial IoT Applications. DDS could be seen as an extended, scalable, high performances data bus. Thanks to the support of Object Management Group (OMG), the world's largest systems software standards organization, DDS has a lot of application around the world. Recently, DDS has been chosen as base middleware for the ROS2 system widely used in robotics.

	Services			
	Acyclic Events	Cyclic Variables	Request /No Response	Request / Response
Deadline	-	Maximum process time	-	-
Durability	Persistent	Volatile	Volatile	Volatile
History	Keep N	Keep last	Keep Last	Keep Last
Latency Budget	Low	Medium	-	-
Ownership	Shared	Shared	Exclusive	Exclusive
Reliability	Reliable	Best effort	Reliable	Reliable
Transport Priority	Highest	High	Low	Lowest

Table 2 - Factory automation traffic mapped to DDS policies **[13]**

Security mechanisms have been added recently to the specification. For instance, authentication in the discovery process ensures that DomainParticipants validate each other's identity, but also Access Control permissions checking ensure that DomainParticipants have the appropriate permissions to exist and match with each other.

B. *SoftwareDefined Network (SDN) and OpenFlow*

An important literature exists on Software Defined Network (SDN) [15][16][17][18]. Only the principles are given in this section. SDN is a networking paradigm where networking control functions, localized before networking elements, are grouped within central entities called SDN controllers. The SDN controllers pilot remotely the networking elements that transfer data. Figure 3 describes the principles of SDN networks. In an SDN architecture, the transmission devices only focus on data transmission, under the supervision of SDN controllers.

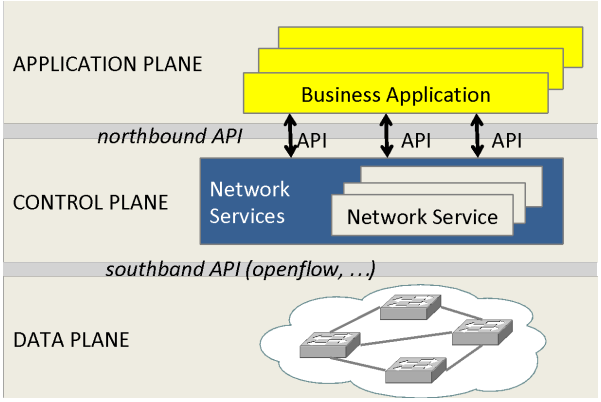

Figure 3 – SDN architecture

The transmission rules associated to each dataflow are sent to the transmission devices by the SDN controller through a South Bound interface. Syntax rules are defined in accordance with the interface specification. The South Bound interface the most used and developed is OpenFlow, proposed and standardized by the industrial consortium Open Networking Foundation (ONF). OpenFlow functionalities support the implementation of complex QoS specifications. They propose a finer network monitoring. Their use reinforces independence from network device providers.

The SDN controller can be compared as a Network Operating System. It centralizes the view and the state of the network, composed of distributed transmission devices. The global view of the network can be shared with applications through a North Bound interface. Applications are informed when networking events and networking changes occur. Moreover, they can transmit their networking requirements, taken into account and managed by the SDN controller.

The SDN approach introduces network flexibility. SDN networks are able to process the application requirement changes and the operational environment changes.

IV. RELATED WORK

The network configuration according to the application requirements principle has been boosted by the dynamic control enabled by SDN concepts. This section reviews such propositions by dividing it into two classes: datacenters and precise frameworks. Datacenter/cloud solutions mainly use SDN to program the network according predefined application profiles. Frameworks mainly focus on solutions that infer the application requirements in order to program the network.

A. *Programmability in datacenters*

Work presented in this section considers the application deployment in datacenters, to speed up the availability of applications. SDN and virtualization are the two technologies used to reserve the underlying network services. These services are embedded within virtual networks built either as overlays to ease to the independence with the network infrastructure, or as infrastructure elements to reinforce the control of virtual links.

The NSX virtualization platform [19] developed by VMWare company creates software communication networks that are integrated in the hypervisor layer, to be managed from a single and automated control point. Virtualized NSX network services include switches, security elements, routing algorithms, firewalls, load balancing algorithms and virtual private networks. The communication application profile is static and explicitly supplied by the administrator. The network service is a virtual network overlay with a flexible QoS defined for dataflow aggregates.

The ACI (Application Centric Infrastructure) platform [20] developed by Cisco company eases the deployment of applications in datacenters. The deployment and the resources required is specified by a group policy language. The network service is a virtual network overlay managed through VXLAN protocol and SDN principles. It integrates functionalities from layer 2 to layer 7. As for NSX, the communication application profile is static and explicitly supplied by the administrator. The network service is a virtual network overlay with QoS defined for dataflow aggregates.

The Netscaler controller [21] developed by Citrix company improves the previous ACI platform. The network service remains a virtual network overlay with QoS defined for dataflow aggregates, but network virtualization is extended with service composition functionalities as dynamic service insertion and service chaining.

The Enabling Application-Driven SDN in the Cloud platform [22] developed by Oracle Solaris and Plurisbus Network companies eases the application deployment and authorizes applications controlling platform resources. Each deployed application is able to control its own virtual private local network (VLAN) through the definition of a Service Level Agreement (SLA). A set of distributed systems linked by Elastic Virtual Switches (EVS) is first defined. It is considered by the Netvisor network hypervisor and realized by using SDN rules. The communication application profile is inferred by the network. The network service is a Virtual Local Area Network that connects the machines where application components are running.

The Application Driven Networking architecture [23] developed by Huawei company, combines the principles of 5G, SDN and Network Functions Virtualization to support the communication requirements of very different applications as video streaming, Internet exchanges, Machine to Machine communications. The traffic profiles of these applications and their associated QoS have very different characteristics in terms of bandwidth, latency and reliability requirements. The system infers the communication application requirements through automatic learning technics and deploys dedicated network slices. It includes its own data and control plane and can include specific protocols and communication mechanisms. The QoS guarantee is flexible and is defined at the level of dataflow aggregates.

As they are specialized for datacenters, these solutions are partially adapted to the applications under consideration in this paper. Specific traffic characteristics of industrial applications,

scalability requirement and QoS guarantees are not considered. These solutions favor coarse grain processing and little flow introspections in order to optimize performance. The following section presents finer solutions that allow better flow treatments but at the expense of less scalability.

B. Precise frameworks

The next group of work mostly applies to campus networks.

ATLAS [24] is a framework that adds to SDN networks the functionality of dynamically and automatically identifying the communicating applications. It uses a supervised automatic learning technic controlled by a set of software agents. By collecting information about active network sockets, the agents communicate with the network plan controller. By merging this information with the network control politic, the controller defines network processing rules to be applied. It applies a differentiated processing for dataflow aggregates where QoS requirements do not change during time.

The NEAT framework [25] adds the application aware functionality to SDN networks. Each computer owns a local NEAT agent. Each application registers to its local agent and specifies its QoS requirement. Local agents interact with the network through its control plan. The applications give explicitly their communication profile to the network. The network associates dataflows to pre-established and pre-calculated data paths.

The MIDAS (MIDdleware Assurance Substrate) framework [26] is another extension of the DDS middleware. At the opposite of the previous work, MIDAS directly embeds the network control in the middleware layer. The MIDAS agents include all the network control functions and have the same capabilities as SDN controllers. By using the Openflow protocol, they can directly program all the communication and networking devices to support dataflow communications. The application communication profiles are inferred by the MIDAS middleware. Then MIDAS selects the network data paths and supplies a guaranteed and potentially dynamic QoS.

The OpenSIP framework [27] reinforces the network awareness of SIP-based applications. It refines the QoS requirements according to the Session Description Protocol used by multimedia applications. All this information is sent and centralized to a Deep Packet Inspection (DPI) program running on the SDN controller. The network then infers the application communication profiles. It associates the application dataflows to the available network data paths.

H. Y. Choi et al. [28] have proposed to reinforce the network awareness of Data Distribution Service (DDS)-based applications. The communication profile is typically derived from the explicit description of the application's requirements from the DDS service. As in the OpenSip framework, their approach uses signalization traffic analysis. The Simple Discovery Protocol (SDP) allows data producers to announce their QoS offered and data consumers to declare their QoS required. This information is sent and analyzed by an inspection program running on the SDN controller. The network selects the network data paths that satisfy the application dataflows requirements with their corresponding priority level.

All these academic works suppose that the communications and requirement needs of each application have to be known in detail during their lifetime. Two main approaches can be identified: In these first approaches (ATLAS, NEAT, and MIDAS), the application that permanently knows its communication requirements is able to control and program the network. In the second approaches, the network is able to infer the application communication profile and can choose and apply the control rules required. Some assumes that this information is explicitly provided to the network by the applications or the users, while the network infers it with the help of DPI, potentially coupled with traffic estimation techniques for others.

As factory application developers are not aware of the traffic characteristics, but know about real-time constraints, we are convinced that a transparent solution is the right solution. In fact, our proposition is close to the H. Y. Choi et al. [28] work. It also relies on DDS to consider timeliness requirements, but adds the data delivery dimension (Table 2) and their evolution. Unicast and multicast are considered for the virtual networks optimization, as traffic characteristics are completely dependent on the number of data receivers.

C. Comparison

Table 3 is a synthetic view of the main characteristics of related work. Three criteria have been retained to compare them. The criterion of application communication profile can be static or dynamic in time. It can be explicitly supplied or inferred by the network. The network service criterion characterizes the Quality of Service (QoS) taken into account by the network and lists the communication elements manipulated by the network. The last criterion lists the management model of each work.

Work	Application communication profile	Underlying network service	Management approach
Datacenters			
NSX	Static Explicitly supplied by administrator	Virtual network overlay Dataflow aggregates Flexible QoS	—
ACI	Static Explicitly supplied by administrator	Virtual network overlay Dataflow aggregates	Group based policy
Netscale	Static Explicitly supplied by administrator	Virtual network overlay Dataflow aggregates	Group based policy

		New services composition functionalities	
Enabling Application Driven SDN	Inferred by the network	Virtual LAN	Service level agreement
Application Driven Networking Architecture	Inferred by the network	Dataflow aggregates Flexible QoS	Machine learning techniques
Frameworks			
ATLAS	Inferred by the network	Dataflow aggregates Constant QoS requirements	Supervised machine learning techniques
NEAT	Explicitly supplied by applications	Dataflows Association with precalculated datapaths	Agents
MIDAS	Inferred by the network	Flexible QoS Datapath selection	Agents
OpenSIP	Inferred by the network	Dataflows Datapath selection	Deep packet inspection
Real-time DDS	Inferred by the network	Dataflows Datapath selection with priority	Signalisation traffic analysis with packet inspection
ADN	Explicitly supplied by applications or Inferred by the network	QoS requirements: rate and latency Unicast and multicast dataflows Automatic resource reservation	Autonomic

Table 3 - Related work analysis and comparison

In summary, our architecture uses the network control principle based on dataflows implemented in most approaches and considers the dynamicity principles as the most advances ones. Determinism and hard QoS constraints are guaranteed by the nature of the defined algorithm. The considered data flows (multicast) and the guaranteed QoS constraints (rate and latency) are consistent with the prod/cons paradigm of the DDS middleware.

V. KEY PRINCIPLES OF OUR PROPOSED ADN ARCHITECTURE

Our main goal is to propose an Application Driven network where the network will be able to supply a flexible and personalized network service that can follow up and adapt to application dynamic needs by reserving instantly and automatically network resources.

One key principle is that the service provided to applications is based on a fine-grained knowledge of instantaneous applications' flows and QoS needs. With the DDS middleware this goes up to identifying the flows of data that are exchanged between each DataWriter and its associated DataReaders. Thanks to a precise knowledge of applications' needs, the right service (that meets exactly the needs) can be provisionned with the optimal set of network resources. Network resource utilization is improved at the cost of scalability.

Another principle is that the network service provided to applications is expressed as a virtual network composed of a set of logical (virtual) end-to-end links (from end host to end host). Each virtual link is either point-to-point or point-to-multipoint and is characterized by a bandwidth requirement and a maximum transfer delay requirement. It assumes that an SDN/OpenFlow enabled network infrastructure is dedicated to the application. A pre-defined slice on network elements exclusively dedicated works also.

As depicted in Figure 4, the network control application "ADN service provisioning" is in charge of provisioning the customized services described above. It is based on a low-level northbound interface (i.e. OpenFlow like).

The last key principle is to build an autonomic "ADN service provisioning" network function. More precisely, we primarily target the self-configuring property and approach the self-healing and self-optimizing properties.

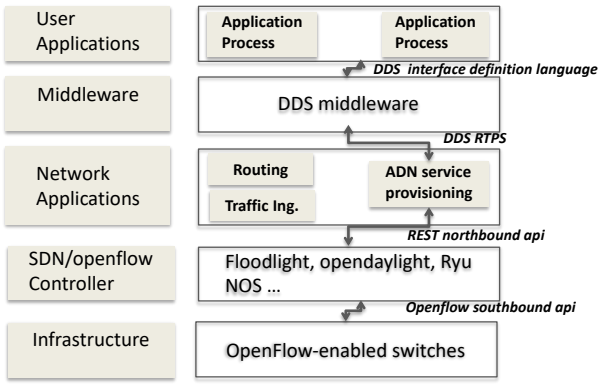

Figure 4 - "ADN service provisioning" network function

VI. General Architecture of the Proposed ADN

This section introduces the functional components of the network control function "ADN service provisioning" that implements our ADN approach. Figure 5 depicts its functional components. These latter are presented hereafter.

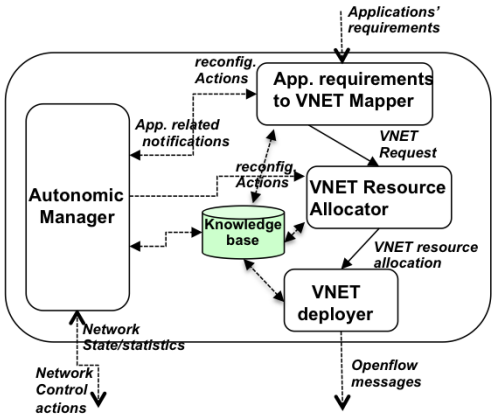

Figure 5 - ADN's functional components

A. Application requirements to Virtual Network Mapper

This component is in charge of mapping application needs to the adequate network service. An application virtual network (VNET) request is composed of a set of end-to-end point-to-point and/or point-to-multipoint logical links, each one with a bandwidth requirement and a maximum transfer delay requirement. This virtual network is the output that the component submits to the "*VNET resource allocator*". Obviously, this component is activated at the beginning of the application but also upon a change in the application's communication profile (change in application's traffic or QoS). In this latter case an update request on the already provisioned virtual network is submitted to the "*VNET resource allocator*".

The data that are exchanged between DDS application processes are typed (defined by the application's topics). Thanks to a "DDS DataWriter", an application process produces a data from a given topic. One or many application processes that have subscribed to this topic can consume and read that data with a "DDS DataReader". So, a precise way to describe the DDS application traffic profile is to reason at the level of a topic. This is the level of detail that we consider for ADN. Next, we denote the data flow belonging to a particular topic as an elementary data flow.

One important function is to characterize all elementary flows that are exchanged by the application: its source "DataWriter" (each topic has one single publisher), its destination(s) "DataReader(s)", its bandwidth and maximum transfer delay. This can be derived as follows:

- In a TCP/IP environment, with DDS, DDS DataReaders and DataWriters are identified with Transport layer addresses: a combination of IP addresses and a port numbers. Transport layer addresses are used to identify the source and the end(s) of elementary flows.
- The bandwidth of an elementary flow is derived from the "Deadline" DDS QoS policy associated to the DataWriter. For periodic production, it specifies the maximum delay between two consecutive data productions and the data size.
- The maximum transfer delay is derived from the "*LatencyBudget"* DDS QoS policy associated to the DataWriter, which specifies the maximum acceptable delay from the time the data is written until the data is inserted in the receiver's application-cache.

As DDS middleware does not offer any function that centralized this information, it has been implemented: A first subcomponent is in charge of capturing and following, on the application DDS domain, the list of topics and their related publishers, subscribers and QoS; Then, the "Elementary flows characterization" subcomponent takes this list as input to derive and characterize the list of elementary flows.

Mapping each elementary flow to an end-to-end virtual link is the most accurate way to describe the network service expected by the application. However, it clearly raises scalability issues. One important aspect is the number of flow table entries that are installed on OpenFlow (OF) switches to support the service. Indeed, current flow-tables, which are often based on fast TCAM memories (Ternary Content Addressable Memory), have a size limited to a few thousands of entries. The "*Flow Aggregation*" subcomponent copes with this problem. It is in charge of computing the final set of flows that describes the expected service by grouping, when feasible, some elementary flows together. The aggregations have generally a cost in term of network resource utilization. This important subcomponent optimizes this tradeoff. In this work, we adopt a simple and intuitive algorithm which groups elementary flows that have the same source and destination end-points.

B. Virtual Network Resource Allocator

This component is at the heart of the proposed ADN. When a virtual network request occurs, it computes the optimal set of physical paths to use in order to support the virtual links with their QoS characteristics. Many optimization criteria can be considered. The ones retained in this work minimize network resource utilization and minimize network elements' load disparities (which contribute to improve the admissibility of subsequent virtual network requests). For the same purpose, *path splitting* (multiple paths to support a virtual link) can be enabled for some requests. A flow can be split into sub flows supported by lower bandwidth links.

Two types of network resources are considered: classically, the bandwidth of links but also the switching resources of nodes, i.e. the number of OpenFlow flow-table entries, group-table entries and meters.

The virtual network resource allocator takes as input the VNET request (the set of virtual links (source and destination(s)) and their bandwidth and delay requirements and then computes the physical network data paths that support each Virtual link, with the required link and switching resources, more precisely: the bandwidth allocation of each virtual link on each physical link as well as the number of flow table entries consumed by each virtual link on each SDN switch. The algorithm is described in section VII.B.

It is worth to note that this component also performs the reallocation or de-allocation of resources in case of an update or a cancellation request.

C. Virtual Network Deployer

The goal of this component is the effective deployment of the virtual network on the OpenFlow network infrastructure. It takes as input the data paths and the associated resources computed by the "VNET Resource allocator", it generates the OpenFlow rules to apply on each OpenFlow switch, and it submits them to the OpenFlow controller via the northbound interface.

Without delving into the details of the algorithm, which is described in section VII.A, the generated OpenFlow rules have the following characteristics: the match field checks for IP addresses and port numbers; They make use of group table entries of type "All" and "Select" to respectively deploy point-to-multipoint links and to apply path splitting. They make use of OpenFlow meters at the source node of each virtual link to enforce the traffic flowing along the link to the assigned bandwidth.

D. Autonomic Manager

The goal of the "*Autonomic manager*" is to instill some of the autonomic properties to the "*ADN service provisioning*" network function. It manages the components described previously by implementing the MAPE (Monitoring, Analysis, Planning, Execution) loop (based on the frameself framework [29]).

Without being exhaustive, some of the important identified situations that the "Autonomic manager" has to react to are described below. On a network topology change (detected from its monitoring of the network), it decides whether network resource re-allocations must be triggered (self-repairing). Network topology changes are notified by the network controller from the reception of states and statistics OpenFlow related messages. All of them are stored in the knowledge database. According to the available network resources and the virtual network request, it tunes the resource allocation (e.g. enable/disable path splitting) and/or the flow aggregation algorithms. Similarly, after virtual network cancellations, it decides to re-compute the allocated resources in order to better distribute the load of network elements (self-optimizing). From its network monitoring, it detects that the rate allocated to a virtual link is not adequate and decides to enable the application traffic estimation (self-configuring).

VII. MAINS ALGORITHMS

In this section, we present the internal algorithms of the proposed "ADN service provisioning" network control function, more precisely, those of the "Virtual Network Deployer" and the "Virtual Network Resource Allocator".

A. The "Virtual Network Deployer" algorithm

The "ADN service provisioning" function is implemented on top of a low-level northbound interface. First, we briefly present some prerequisites on the OpenFlow protocol before detailing the algorithm. To simplify, the presented algorithm does not address the deployment of point-to-multipoint link with path splitting enabled.

1) Some prerequisites on Openflow

An OpenFlow switch embeds at least one flow table, a group table and a meter table. The OpenFlow controller relies on OpenFlow modification messages to fill these tables. Three types of messages are distinguished by the OpenFlow protocol. They are described hereafter.

OpenFlow Flow Modification Messages (ofp_flow_mod) are used by the controller to insert, delete or update one flow entry into a flow table of a switch. Each flow entry contains a match field and a set of instructions that are triggered when a packet matches the entry. These instructions result in changes to the packet action set. There are multiple types of instructions, among which, the "write-actions" and "meter", which are used in the proposed algorithm. "Write-Actions" instruction gathers a list of actions to add to the current "Action-Set" of the matching packet. The Meter instruction guides the matching packet to the specified meter.

OpenFlow Group Modification Messages (ofp_group_mod) are sent by the controller to insert, delete or update a group entry into the group table of a switch. Each group entry has a group id; it is typically used by a flow table entry as a reference to the group. A group entry has a list of action buckets. Depending on the group type, the actions in one or more action buckets are applied to packets sent to the group. Two types have been retained in this work:

- "Select": used for load sharing. Each bucket has a weight, which is used to choose the bucket that applies to an arriving packet.
- "All": used to perform multicast or broadcast forwarding. The packet is cloned for each bucket.

OpenFlow Meter Modification Messages (ofp_meter_mod) are sent by the controller to insert, delete or update a meter entry into the meter table of a switch. A meter entry is identified by a meter id and is composed of one or more meter bands. Each meter band specifies a target rate for that band and the way packets are processed when that rate is exceeded: it is either dropped (for a meter band of type "Drop") or remarked (for a meter band of type "DSCP remark").

A last point concerns the order with which OpenFlow Flow Modification Messages are sent to a switch. Any reference must be set on a flow table entry, meter or group already created. This implies that the processing of the OpenFlow meter (or group) modification message at the switch must precede the OpenFlow flow modification message. As described below, our algorithm arranges the transmission of modification messages to respect this constraint.

2) The proposed algorithm

The goal of the algorithm is to build the list of OpenFlow Modification Messages to deliver to each network node involved in the support of the virtual links that compose the Virtual Network. In fact, three lists of messages are computed for each node: 1) the list of OF Meter Modification Messages, 2) the list of OF Group Modification Messages and 3) the list of OF Flow Modification Messages. Once computed, the algorithm instructs the controller to convey them to node in the order specified above. Our algorithm uses (without being a necessity) the bundle mechanism introduced in OF version 1.4, for message grouping and ordering as well as, to store and pre-validate them on each node before a global confirmation across multiple nodes. The main steps of the algorithm are:

For each node *i* crossed by the virtual network [line 2], and for each virtual link *k* [line 3]:

- If the node *i* is the source node of virtual link *k*, then it inserts an OpenFlow meter into meterModMessageList and keeps

the meterID for later use (when building the flow modification message of the flow table entry that refers to the meter). [line 4 – line 8]

- If the virtual link *k* is split at node *i*, then it inserts a group into groupModMessageList and it inserts a flow rule into flowModMessageList with that group as action of "Write-Actions" instruction. If the node *i* is the source of virtual link k, a meter instruction is also added [line 20 – line 34]. If *k* is not split at node *i*, then it inserts a flow into flowModMessageList with a simple output port action in the "Write-Actions" instruction, and eventually, a meter instruction if the node is the source of virtual link k [line 10 – line 19].
- It transmits successively the meterModMessageList, groupModMessageList, flowModMessageList to node *i's* bundle. This latter is configured with the ordered flag set, to request that the messages of the bundle are processed in the order of arrivals. [line 37-End].

VNET Deployer Algorithm : (V, E, K)

ut :
is the set of nodes (Openflow Switches). Each node i processes and forwards data to another node j via a port noted p_{ij}
s the set of links. The link between the nodes i and j is noted (i,j).
s the set of virtual links. Each virtual link k is characterized by: a source node s_k element of V, a bandwith b_k, and a set of destination nodes T_k part of V except s_k
a 4-dimensional dictionary of integer noted $f_k^t(i,j)$, representing bandwidth allocated at link (i,j) to packets of virtual link k, that are flowing from the source node s_k to the destination note t.
a 3-dimensional dictionary of integer noted $f_k(i,j)$, representing bandwidth allocated at link (i,j) to packets of virtual link k.
is the set of nodes crossed by at least one virtual link.

Var : match : ofp_match; group_mod : ofp_group_mod; nextHops a set of nodes; flowModMessageList : ofp_flow_mod[]; groupModMessageList : ofp_group_mod[]; meterModMessageList : ofp_meter_mod[]; groupID, meterID, bundleID : integer;

```
1   begin
2   for each i in V' do
3       for each unicast link k in K
4         if(i = sk) then
5           meterID←get_meter_id(i,k)
6           insert_into (meterModMessageList, {id =meterID, bands [0] ={rate = bk,
7   type = drop}})
8         end if
9         nextHops←computeNextHops(i, k,V', E, f)
10        if (|nextHops| = 1) then
11          for each j in nextHops do
12            if(i = sk) then
13              insert_into (flowModMessageList, { match=create_match(),
14              instructions ={meter: meterID, write-actions: output.port=Pij})
15            else
16              insert_into (flowModMessageList, { match=create_match(),
17              instructions ={write-actions: output.port =Pij}})
18            end if
19          end for
20        else
21          groupID←get_group_id(i,k)
22          group_mod.id ←groupID; group_mod.type←select
23          for each j in nextHops do
24            group_mod.buckets[j] ← {weight =fk(i,j), actions ={ouput.port=Pij} }
25          end for
26          insert_into (groupModList, group_mod)
27          if (i = sk) then
28            insert_into (flowModMessageList, {match=create_match(),
29            instructions = {meter: meterID, write-actions: group.id=groupID}})
30          else
31            insert_into (flowModMessageList, match =create_match(),
32            instructions ={write-actions: group.id =groupID}})
33          end if
34        end if
35      end for
36
37      bundleID← get_bundle_id(i)
38      OFPBCT_OPEN_REQUEST {id =bundleID, flags =ordered}
39      for each msg in meterModMessageList do
40        OFPT_BUNDLE_ADD_MESSAGE {id = bundleID, message = msg}
41      end for
42      for each msg in groupModMessageList do
43        OFPT_BUNDLE_ADD_MESSAGE {id = bundleID, message = msg}
44      end for
45      for each message in flowModMessageList do
46        OFPT_BUNDLE_ADD_MESSAGE {id =bundleID, message = msg}
47      end for
48      reset (meterModMessageList); reset (groupModMessageList); reset
49      (flowModMessageList)
50  end for
51  for each i in V' do
52      OFPBCT_CLOSE_REQUEST {id = get_bundle_id(i)}
53      OFPBCT_COMMIT_REQUEST {id =get_bundle_id(i)}
54  end for
    end
```

B. VNET Resource Allocator

This section describes the Integer Linear Programming (ILP) formulation proposed to solve the resource allocations for the virtual network. Virtual network Requests arrive and are processed in sequence with no information on future requests. For each request, the output is the set of routes (with the bandwidth allocations at each supporting physical link and the number of flow table, meter table and group table entries at each crossed node) that support each of the virtual links composing the request. This algorithm extends our previous work [30] by considering meter tables and group tables (in addition to the flow tables) as network resources to assign. For meter tables, things are quite simple since OpenFlow meters are only activated on the source nodes of the virtual links. The algorithm simply checks that meter table entries are still available at source nodes. Actually, switch memories (TCAM) are very small and can only support a small number of rules, either in terms of flows, meters or groups numbers. If so, the algorithm assigns a meter table entry on each of these nodes and proceeds with the other resources as described below. If not, the request cannot be honored. Our algorithm is the first that takes into consideration all of these constraints (noted l, n, u hereafter).

1) Physical Network Model

The physical network is modelled by a bidirectional graph $G = (V, E)$ where $V(|V|)$ is the set of physical nodes (SDN switches) and $E(|E|, E \subseteq V \times V)$ the set of physical links which operate in full-duplex mode. To each node $i \in V$, is associated a switching capacity U_i, which is the maximum number of entries (i.e. size limit) of its flow table. The current size of node i flow table is denoted by U_i'. Similarly, N_i and N_i' denote respectively the maximum and the current size of the group table of switch i. Each Link $(i, j), i, j \in V$ *i*s weighted by its bandwidth B_{ij} and its propagation delay D_{ij}. Links are assumed to have the same characteristics in both directions, i.e. $B_{ij} = B_{ji}$ *a*nd $D_{ij} = D_{ji}$. The bandwidth that is currently assigned at link (i, j), by already admitted virtual links is denoted by B_{ij}'.

2) Virtual Network Requests Model

A virtual network request is composed of a set of K virtual links. Each virtual link k is characterised by:

- a source node $s_k \in V$, and a set of destination nodes $T_k \subseteq V - \{s_k\}$ (when $|T_k| = 1$, the virtual link is point-to-point, otherwise it is point-to-multipoint);

- a bandwidth requirement of $b_k \in N$, a maximum transfer delay of $d_k \in N$ and a maximum packet size of p_k.

3) Resource-related assignment variables

Basic assignment variables are related to a specific destination of a virtual link. In our model, we distinguish the following variables:

- $f_k^t(i,j)$ is an integer variable that represents the bandwidth allocated at link (i,j) to the packets of virtual link k that are flowing from the source node s_k to a destination node t. More generally, $f_k(i,j)$ refers to the amount of bandwidth used on link (i,j) by the virtual link k. It is set to the maximum of $f_k^t(i,j)$for all $k \in K$.

- $l_k(i)$ is an integer variable that specifies the number of entries that are installed in node i flow table to support virtual link k with the assumption that all entries consume the same amount of resources regardless of the complexity of the match operation and the related instructions to perform. A flow table entry is added if at least one of node i port is supporting traffic from k (equations 1).

$$\forall k \in K, i \in V: \quad 0 \le l_k(i) \le 1 \qquad (1.a)$$
$$and \quad \forall j \in V, \forall (j,i) \in E: \quad g_k(j,i) \le l_k(i) \qquad (1.b)$$
$$l_k(i) \le \sum_{j \in V\ (j,i) \in E} g_k(j,i) \qquad (1.c)$$

where $g_k(j,i)$ is an intermediate boolean variable that indicates if some bandwidth from link (j,i) is assigned to virtual link k or not. It is derived from another set of more focused intermediate variables $g_k^t(j,i)$ that reflects whether the flow of packets of virtual link k destined to t is supported by the physical link (j,i) (i.e.$g_k^t(j,i) = 0$ $if\ f_k^t(j,i) = 0$ and $g_k^t(j,i) = 1$ otherwise).

- $n_k(i)$ is an integer variable that specifies the number of group table entries assigned to k at node i. As described in section VI.A, a group table entry is added when splitting or when duplicating packets (for point-to-multipoint links).

- f_{max} is the maximum link utilization (when considering all network links) after request acceptance.

- u_{max}: is the maximum flow table utilization (when considering all network nodes) after request acceptance.

All the above presented variables (except the two last) define the data paths that support the VNET to embed, with the required link/bandwidth resources and switching resources (flow/meter/Group table entries). f_{max} (respectively u_{max}) which are minimized in the objective function are respectively exploited to limit the load disparity between network links (resp. nodes).

4) Problem Constraints

The constraints on bandwidth allocations are described in equations 2 to 8. Equation 2 reflects the linearization of the Max operator applied to variables $f_k^t(i,j)$ to get $f_k(i,j)$. Equations 3 and 4 have a similar purpose and focus respectively on f_{max} and u_{max}, which are minimized by the objective function (as explained below).

$$\forall k \in K, \forall (i,j) \in E, \forall t \in T_k: \quad f_k^t(i,j) \le f_k(i,j) \qquad (2)$$

$$\forall (i,j) \in E: \quad \frac{1}{B_{ij}} * \left(B'_{ij} + \sum_{k \in K} f_k(i,j) \right) \le f_{max} \qquad (3)$$

$$\forall i \in V: \quad \frac{1}{U_i} * \left(U'_i + \sum_{k \in K} l_k(i) \right) \le u_{max} \qquad (4)$$

Equation 5 ensures that the bandwidth assigned to each virtual link k at link (i,j) does not exceed the remaining bandwidth. Equation 6 is the usual flow conservation constraints.

$$\forall (i,j) \in E: \sum_{k \in K} f_k(i,j) \le B_{ij} - B'_{ij} \qquad (5)$$

$$\forall k \in K, \forall t \in T_k, \forall i \in V:$$

$$\sum_{k \in K} (f_k^t(i,j) - f_k^t(j,i)) = \begin{cases} b_k & if\ i = s_k \\ -b_k & if\ i = t \\ 0 & else \end{cases} \qquad (6)$$

Equation 7 is a channeling constraint between integer and boolean variables: $f_k(i,j)$ and $g_k(i,j)$. It also constrains the virtual link k's bandwidth assignment at a physical link to the requested bandwidth b_k. Equation 8 constrains the bandwidth that is assigned to the flow of packets destined to a specific virtual link's end-point. The inequality on the right side ensures that the bandwidth requirement of the virtual link is never exceeded. The inequality on the left side directs path-splitting and avoids the multiplication of splits with low bandwidth allocations. Indeed, if active, path splitting is feasible only if the bandwidth allocated to the splits respects a minimum threshold b_k^{min}. In practice, b_k^{min} is a ratio of b_k, $b_k^{min} = PS_{ratio} * b_k$, with $PS_{ratio} \in [1,0]$.

$$\forall k \in K, \forall (i,j) \in E:$$
$$g_k(i,j) \le f_k(i,j)\ and\ f_k(i,j) \le b_k * g_k(i,j) \qquad (7)$$

$$\forall k \in K, \forall (i,j) \in E:$$
$$b_k^{min} * g_k^t(i,j) \le f_k^t(i,j)\ and\ f_k^t(i,j) \le b_k * g_k^t(i,j) \qquad (8)$$

The constraints related to switching resource allocations are described in equations 9 and 10. Equation 9 simply ensures that with the addition of flow table entries needed by the virtual links composing the request, the size of network nodes' flow tables remains below their maximum size.

$$\forall i \in V: \sum_{k \in K} l_k(i) \le U_i - U'_i \qquad (9)$$

Equations 10 constrain the allocations of group table entries. Equation 10.b applies when no group entries are needed for the virtual link k at node i (it neither traverses i nor requires a flow split or packet duplication). Equation 10.c applies when a group

entry is needed. Finally, equation 10.d simply ensures that the addition of group entries that are needed by the virtual links respect the maximum size of all the group tables.

$$\forall k \in K, \forall i \in V : \; 0 \leq n_k(i) \leq 1 \qquad (10.a)$$

$$\forall k \in K, \forall (i,j) \in E:$$

$$n_k(i) \leq \left(\sum_{k \in K \; (i,j) \in E} g_k(i,j) \right) - g_k(i,j) \qquad (10.b)$$

$$\forall k \in K, \forall (i,j_1), (i,j_2) \in E, j_1 \neq j_2 :$$

$$n_k(i) \geq g_k(i,j_1) + g_k(i,j_2) - 1 \qquad (10.c)$$

$$\forall i \in V : \; \sum_{k \in K} n_k(i) \; \leq N_i - N_i' \qquad (10.d)$$

5) Objective function

The objective function aims at minimizing link and node resource consumption but also at distributing the consumed resources among nodes and links in order to reduce the creation of bottlenecks. Both contribute to improve the admissibility of forthcoming requests. As shown in expression 11, it consists of four components, each weighted with a parameter that controls the impact of the component on the resolution process. The first two ones concern bandwidth allocations and the last two ones concern flow table entries allocations.

Minimize

$$\alpha_1 * \frac{1}{|E|} * \sum_{(i,j) \in E} \left(\frac{1}{B_{ij}} * \left(B'_{ij} + \sum_{k \in K} f_k(i,j) \right) \right) + \alpha_2 * f_{max} + \beta_1 * \frac{1}{|V|} * \sum_{i \in V} \left(\frac{1}{U_i} * \left(U'_i + \sum_{k \in K} l_k(i) \right) \right) + \beta_2 * u_{max} \qquad (11)$$

VIII. Implementation and Tests of Feasability

This section describes the proof-of-concept implementation of the proposed ADN.

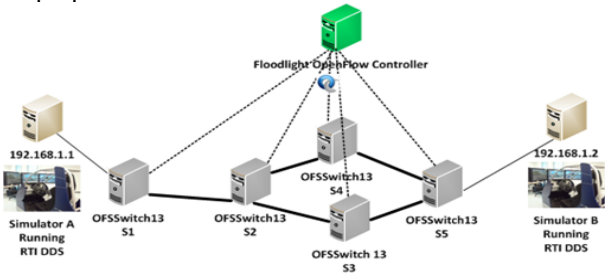

Figure 6 - Testbed environment

A. Considered Platform

Figure 6 depicts the OpenFlow network infrastructure that supports our ADN implementation. It consists of five Linux devices running OFSoftSwitch13 [18] software switch. They are interconnected using GRE Tunnel over Ethernet physical links. These links own the following performances: 100Mbps of bandwidth and 10µs of propagation delay, except the link between S1 and S2, which has 1Gbps of bandwidth. The Floodlight SDN controller platform [17] is used as the OpenFlow controller. One of the interesting features of Floodlight V.1.0 and OFSoftSwitch13 is their full support for OF 1.3. Meters and groups of type "all" and "select". Moreover, an experimental support for OF 1.4 is possible, including bundles with atomic modification features. Also, Floodlight provides common network functionalities such as topology discovery. The "RTIConnextDDS" DDS implementation is used.

B. Implementation

The following components have been implemented. The "Application requirements to VNET Mapper", which implements all the subcomponents previously described. It continuously captures, for a given application, its active topics and their associated publishers and subscribers (with their location) as well as their QoS demands. Then, it establishes the list of elementary flows, and finally performs flow aggregation based on basic strategies. The "Virtual Network Resource allocator" implements the algorithm presented in section VII.B using concert technologies C++ as the modeling layer and IBM CPLEX 12.6 as solver. The "Virtual Network Deployer" implements the algorithm of section VII.A as an application module that interfaces with the Floodlight Java-based API.

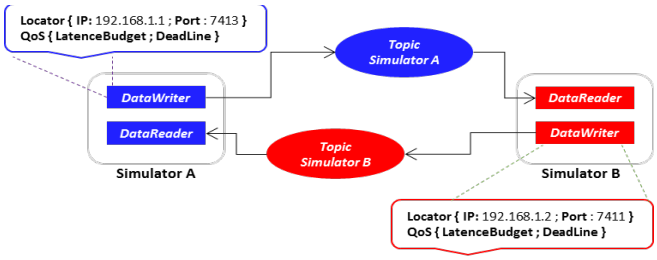

Figure 7 – DDS view of the considered application

C. Considered DDS Application

A demonstrative application with stringent and dynamic requirements has been chosen. We consider a distributed interactive simulation for vehicle driver training that involves two networked driving simulators (called simulator A and B) that evolve in a shared virtual world. Each simulator needs to get state information (position, speed, etc.) from nearby simulators. The closer are the simulator, the more frequent state information must be exchanged and the more stringent the required delay to get these states is. Movement of driving simulators (in the virtual world) brings 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. Figure 7 describes the application from the DDS perspective with the topics related to the simulators' state information, the DDS QoS policies, etc.

This application was chosen as representative of the new requirements that can be found in the use of augmented reality, smart monitoring, in a framework compatible with Industrial

Ethernet solutions. The manipulated data are cyclic, with variable frequencies and generate high flow rates with stringent delay requirements.

D. *Illustrative results*

We consider the scenario presented in Figure 8, where "Simulator B" starts moving at t_0 towards the static "Simulator A". We assume that at time t1, the QoS requirements related to the distribution of simulators' state information must be changed to a larger bandwidth and a stricter transfer delay. We focus below on how our proposed ADN provisions and then adjusts the network service that it provides to the application, to respond to the new requirements.

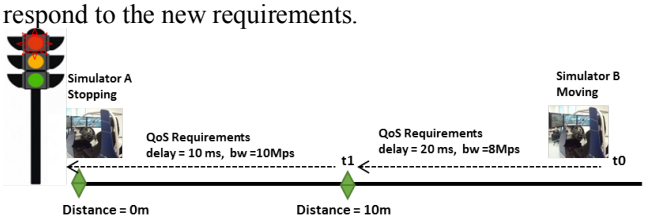

Figure 8 Considered application scenario

Figure 9 shows the application requirements that were captured by the "*Application Requirements to VNET Mapper*" from time t_0 to t_1 (exclusive). The DataWriter and DataReaders of the two topics (simulator A and simulator B) are identified as well as their DDS related QoS. Figure 10 also shows the derived elementary flows: one from node S1 to S5 and another from node S5 to S1 with the same QoS requirements (d=20ms, bw=8Mbps). These two flows form the virtual network to provision for the application. The result of the resource allocations on the OpenFlow network at time t0 is presented in Figure 10. They were obtained with $\alpha_1 = \beta_1 = 1$; $\alpha_2 = \beta_2 = 2$ and with path splitting enabled ($PS_{ratio} = 0.3$). This means that minimizing link load disparity is the preferred optimization criterion. The 8 Mbps requirements to S5 are reserved on the S1-S2 link, then the request is splitted on a 4 Mpbs reservation through S3, and 4 Mpbs through S4.

The mobility of the simulators (in the virtual world) brings dynamicity in the data flows that are delivered/consumed by each simulator and on their associated QoS. When simulators get closer, requirements are tighter. Figure 11 describes the new application requirements captured at time t_1, the computed update of the characteristics of the virtual links. 2Mbps more are required on both directions. The previous virtual links are updated consequently. Finally, Figure 12 describes the corresponding resource allocations starting from time t_1.

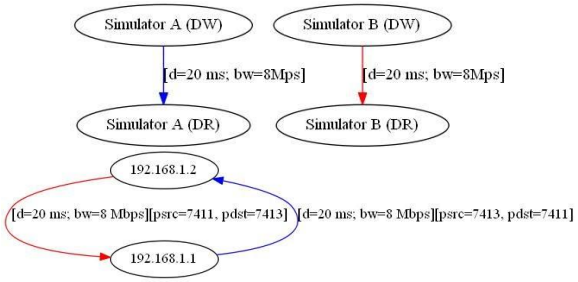

Figure 9 – Output of the "Application Requirements to VNET Mapper" at time t_0

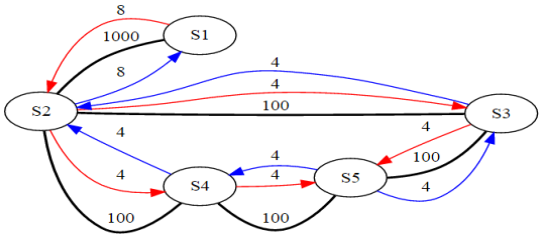

Figure 10 – Output of the "VNET Resource Allocator" at time t_0

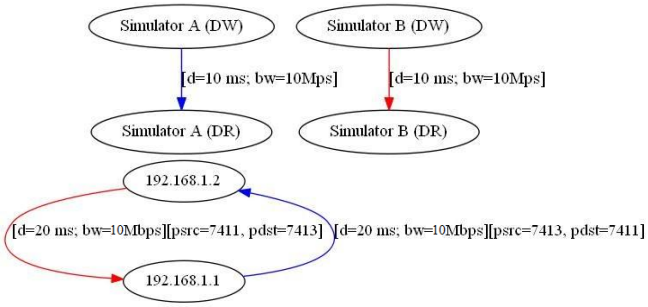

Figure 11 - Output of the "Application Requirements to VNET Mapper" at time t_1

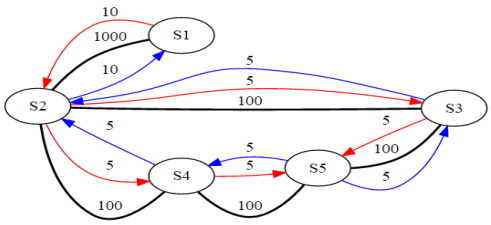

Figure 12 - Output of the "VNET Resource Allocator" at time t_1

In the next section, a performance evaluation with more complex scenarios is presented.

IX. PERFORMANCE ANALYSIS

For better highlighting our algorithm performances, we extend the former scenario to groups of DDS simulators supported by more networking devices like future large-scale factory networks will be. In this case, the topology of a real campus network (Figure 13) with 31 nodes and 55 links (with 100Mbps and 1Gbps) is considered as the new SDN substrate. The network has been extended by adding MiniNet emulated openflow nodes and virtual links.

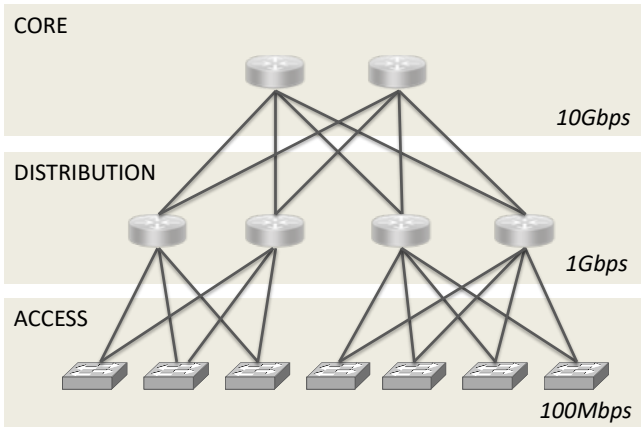

Figure 13 - campus network

Evaluations of the resource allocation algorithm were realized to assess its general performance and benefits in comparison to some Shortest Path (SP) heuristics. The results presented below were based on the hierarchical campus network topology cited above with the flow table and group table sizes respectively set to 2000 and 512 entries. VNET requests are assumed to arrive according to a Poisson process with an arrival rate range from 4 to 10 requests per 100 Unit of Time (UT). Each VNET request is composed of a number of virtual links that is randomly chosen between 1 and 4. Each virtual link has a number of destinations that is randomly chosen between 1 and 4 and has a bandwidth requirement randomly chosen between 1 and 3 Mbps. Once a VNET request is accepted it lasts till the end of the experiment, which is set to 10000 UT. Path Splitting (PS) is activated with a PS_{ratio} set to 0.3. The simulation setups have been chosen to stress the network while conforming to the switch specifications (i.e. flow table size <2000 OF rules).

The considered SP heuristic is defined as follows. A cost function assigns a cost to each physical link that is inversely proportional to its current available capacity. For each couple of end-points that belongs to a virtual link, the physical path with the minimum cost is chosen. If the bandwidth available for the chosen path is below the bandwidth required by the virtual link, the corresponding request is rejected.

The algorithm performances have been measured according to three criteria: the request acceptance rate, the use of link and nodes rates, the convergence duration. The last two criteria, that to our opinion are the most important ones to characterize the usability of our proposal, have been compared with the shortest path heuristics.

The following performance metrics are computed during simulation for performance analysis purposes:

Acceptance rate: the percentage of successful virtual links requests out of all the requests that arrived during the simulation time;

Convergence duration: the time needed by our algorithm to compute the optimal allocations associated to a virtual links request. The average convergence time and the maximum convergence time are computed over the number of successful requests;

Link utilization: the percentage of assigned bandwidth at a given link. Minimal and Maximal values are also given.

The following subsections presents a synthesis of the main results obtained. A more in-depth performance analysis with more parameters and values has been presented in [31].

A. ADN resource allocation algorithm performances

Figure 14 presents the link use rate of the ADN algorithm. Our experiments show that the average link utilization is between 60 and 80% at some backbone links, we observe, at the end of the experiments, that more than 95% of their capacity has been allocated. This maximal use of some links strongly influences the request acceptance rate. To avoid its important decrease, the path splitting ratio chosen can be increased.

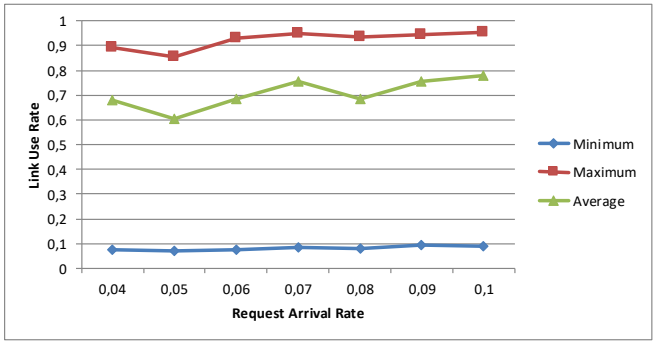

Figure 14 – Link use rate

Figure 15 presents the node use rate of the ADN algorithm. The load required to compute the ADN algorithm remains at a low level: less than 20% of the node capacity is required.

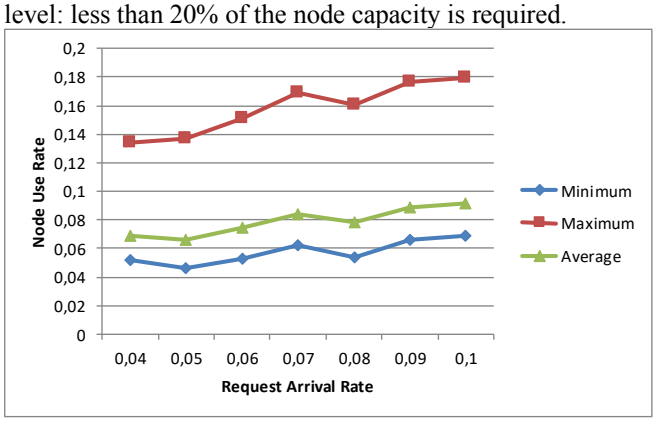

Figure 15 – Node use rate

Figure 16 presents the convergence duration of the ADN algorithm to compute the optimal allocations associated to a VNET request. For our algorithm, the convergence duration remains at acceptable levels: on average below 60ms and a longest convergence duration of 550ms.

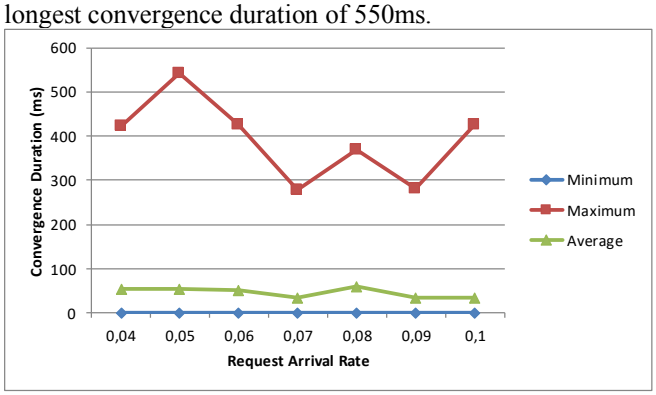

Figure 16 – Convergence duration

A. ADN resource allocation algorithm and shortest path heuristics comparison

Figure 17 describes the requests acceptance rate as a function of the request arrival rate. Under this high load, it clearly shows that our algorithm achieves an acceptance rate significantly greater than the heuristic.

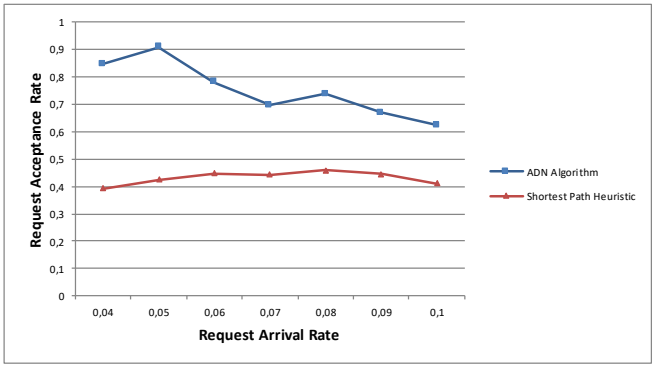

Figure 17 – Requests Acceptance Rate

Figure 18 presents the average duration needed by our algorithm and the SP heuristic to compute the optimal allocations associated to a VNET request. As SP heuristic is faster than the ADN algorithm, the results obtained balance the advantage obtained by a better acceptance rate. Our algorithm requires more time to obtain better results. However, to our opinion, these durations remain acceptable (less than 60ms on average).

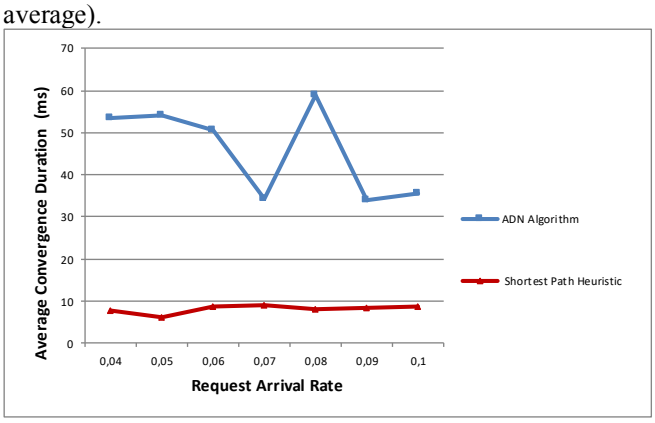

Figure 18 – Convergence duration

X. Conclusion and Future Work

This work proposes a SDN Based Application Driven Network that is able to provide QoS enabled data-paths on an application flow basis. This allows providing tailored network services to applications while using efficiently network resources (even when application requirements are dynamic). This detailed consideration of applications communication profile has a cost in terms of scalability. Clearly, the intention is not to apply the proposed ADN to any application and in any context. It rather targets real-time or business critical applications in a private infrastructure, industry or campus networks, where scalability is not the primary concern. The proposed ADN was implemented and applied to prove its feasibility. The DDS/SDN based approach makes it a turnkey solution and only a little effort is required to configure it. A server with the ADN software is added to the network and each openflow switch has to refer it as controller. The server has to subscribe to all of the DDS topics.

Perspectives of this work mainly concern the extension to multidomain networks. Several works have started on this topic as ONF TAPI [32] or IETF ALTO [33]. Our architecture should rely on such descriptions to negotiate multidomain virtual links.

Acknowledgment

This work has been partially funded by the French National Research Agency (ANR), the French Defense Agency (DGA) under the project ANR DGA ADN (ANR-13-ASTR-0024).

References

[1] V. Alcácer and V. Cruz-Machado, Scanning the Industry 4.0: A Literature Review on Technologies for Manufacturing Systems, Engineering Science and Technology International Journal, Elsevier, Vol. 22, Issue 3, pages 899-919, June 2019.

[2] W. He and L. D. Xu. Integration of Distributed Enterprise Applications: A Survey. IEEE Transactions on Industrial Informatics, vol. 10, No 1, pp. 35–42, Feb. 2014.

[3] J. Follows and D. Straeten, Application-Driven Networking: Concepts and Architecture for Policy-Based Systems , IBM Red book, dec.1999.

[4] A. Georgi, R. G. Budich, Y. Meeres, R. Sperber, and H. Hérenger, An integrated SDN architecture for application driven networking, International Journal on Advances in Systems and Measurements, vol. 7, pp. 103–114, 2014.

[5] T. Zinner, M. Jarschel, A. Blenk, F. Wamser and W. Kellerer. Dynamic application-aware resource management using Software-Defined Networking: Implementation prospects and challenges. IEEE/IFIP Network Operations and Management Symposium (NOMS), Krakow, Poland, 5-9 may 2014.

[6] H. Mekky, F. Hao, S. Mukherjee, Z. L. Zhang and T.V. Lakshman. Application-aware data plane processing in SDN. Proceedings of the ACM: third workshop on Hot topics in software defined networking (HotSDN '14), Chicago, USA, Aug. 22 2014, pages 13-18.

[7] M. Savi and D. Siracusa. Application-aware service provisioning and restoration in SDN-based multi-layer transport networks. Optical Switching and Networking, Volume 30, Nov. 2018, Pages 71-84, Elsevier.

[8] M. Wollschlaeger, T. Sauter and J. Jasperneite, The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0 in IEEE Industrial Electronics Magazine, vol. 11, no. 1, pp. 17-27, March 2017.

[9] GSMA 2018. Network Slicing – Use Cases Requirements. https://www.gsma.com/futurenetworks/wp-content/uploads/2018/07/Network-Slicing-Use-Case-Requirements-fixed.pdf/ accessed Jan. 2020.

[10] 3GPP TS 22.804: Study on Communication for Automation in Vertical Domains

[11] 3GPP TS 38.300: NextGen Radio Access Network (NG-RAN); Overall description; Stage 2

[12] Object Management Group, Data-Distributed Service for Real-Time Systems, OMG, version 1.4. Sept. 2014.

[13] I. Calvo, F. Pérez, O. G. de Albeniz and I. Etxeberria-Agiriano, "Towards a OMG DDS communication backbone for factory automation applications," ETFA2011, Toulouse, 2011, pp. 1-4.

[14] R. S. Auliva, R. Sheu, D. Liang and W. Wang, IIoT Testbed: A DDS-Based Emulation Tool for Industrial IoT Applications, 2018 International Conference on System Science and Engineering (ICSSE), New Taipei, 2018, pp. 1-4.

[15] D. S. Rana, S. A. Dhondiyal and S. K. Chamoli, Software Defined Networking (SDN) Challenges, issues and Solution, International Journal of Computer Sciences and Engineering, Vol 7, Issue 1, pages 884-889, January 2019.

[16] Open Networking Foundation, OpenFlow Notification Framework, ONF TS 014, Version 1.0, October 2013,

[17] B. S. Networks, Floodlight openflow controller,. Version 1.0,. 2013, [Online]. Available: http://www.projectfloodlight.org/floodlight/

[18] CPqD, OFSoftSwitch 13, [Online]. Available: http://cpqd.github.io/ofsoftswitch13

[19] VMware® NSX Network Virtualization Design Guide. Available at https://www.vmware.com/products/nsx.html, accessed Dec. 2019.

[20] Cisco. Application Centric Infrastructure (ACI). Available at https://www.cisco.com/c/en_uk/solutions/data-center-virtualization/application-centric-infrastructure/index.html, accessed Dec. 2019.

[21] Citrix NetScaler. Available at https://en.wikipedia.org/wiki/NetScaler, accessed Dec. 2019.

[22] Oracle Solaris 11 and Pluribus Networks - Enabling Application-Driven SDN in the Cloud'. Available at https://blogs.oracle.com/openomics/oracle-solaris-11-and-pluribus-application-awareness-in-the-cloud-v2', accessed Dec. 2019.

[23] Y. Wang, D. Lin, C. Li, J. Zhang, P. Liu, C. Hu and G. Zhang. Application Driven Network : Providing On-Demand Services for Applications. ACM SIGCOMM Conference, SIGCOMM '16, pages 617–618, New York, NY, USA, 2016.

[24] Z. Ayyub Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt and G. Noubir. Application-awareness in SDN. SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pages 487–488, August 2013.

[25] R. Santos, Z. Bozakov, S. Mangiante, A. Brunstrom and A. Kassler. A NEAT framework for application-awareness in SDN environments. IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), nov. 2017.

[26] A. L. King, S. Chen and I. Lee. The MIDdleware Assurance Substrate : Enabling Strong Real-Time Guarantees in Open Systems with OpenFlow. IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, pages 133–140, June 2014.

[27] A. R. Montazerolghaem, M. H. Y. Moghaddam and A. Leon-Garcia. OpenSIP : Toward Software-Defined SIP Networking. IEEE Trans. Network and Service Management vol. 15 no. 1, pages 184-199, 2018.

[28] H. Y. Choi, A. L. King and I. Lee. Making DDS really real-time with Open-Flow. International Conference on Embedded Software (EMSOFT), pages 1–10, Oct 2016.

[29] M. Ben Alaya and T. Monteil, FRAMESELF: A Generic Context-Aware Autonomic Framework for Self-Management of Distributed Systems, in IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012.

[30] M. Capelle, S. Abdellatif, M.J. Huguet and P. Berthou, Online Virtual Links Resource Allocation in Software-Defined Networks, IFIP Networking 2015, Toulouse, France, 20 – 22 May 2015.

[31] A. F. Simo Tegeu. Towards networks guided by and for highly dynamic applications. PhD thesis, University of Toulouse, July 2018.

[32] V. Lopez, R. Vilalta, V. Uceda, A. Mayoral, R. Casellas, R. Martínez, R. Muñoz and J. P. Fernandez Palacios. Transport API: A solution for SDN in carriers networks, 42nd European Conference on Optical Communication (ECOC), September 2016.

[33] D. A. Lachos Perez, C. Esteve Rothenberg and R. Szabo , Broker-assisted Multi-domain Network Service Orchestration, IEEE Wireless Communications and Networking Conference (WCNC), April 2018.