Scalable Software Defined Monitoring for Service Provider DevOps

Wolfgang John, Catalin Meirosu
Ericsson Research, Sweden
Email: first.lastname@ericsson.com

Bertrand Pechenot, Pontus Sköldström
ACREO, Sweden
Email: {berpec,ponsko}@acreo.se

Per Kreuger, Rebecca Steinert
SICS, Sweden
Email: {piak,rebste}@sics.se

***Abstract*—Technology trends such as Cloud, SDN, and NFV are transforming the telecommunications business, promising higher service flexibility and faster deployment times. They also allow for increased programmability of the infrastructure layers. We propose to split selected monitoring control functionality onto node-local control planes, thereby taking advantage of processing capabilities on programmable nodes. Our software defined monitoring approach provides telecom operators with a way to handle the tradeoff between high-granular monitoring information versus network and computation loads at central control and management layers. To illustrate the concept, a link rate monitoring function is implemented using node-local control plane components. Furthermore, we introduce a messaging bus for simple and flexible communication between monitoring function components as well as control and management systems. We investigate scalability gains with a numerical analysis, demonstrating that our approach would generate thousandfold less monitoring traffic while providing similar information granularity as a naive SNMP implementation or an OpenFlow approach.**

I. INTRODUCTION

Recently, there have been many innovations in computer networking. SDN breaks the traditional all-in-one network equipment by separating control from data plane. Network virtualization techniques allow service providers to slice infrastructure resources, enabling a flexible deployment of new network technologies. The evolution of Cloud computing makes it possible to merge physically distributed facilities into a united logical resource. Taken together, the emerging software defined "everything" (networks, compute infrastructure, etc.) paradigm promises a high level of flexibility and programmability, allowing telecom operators to elastically reallocate the infrastructure resources of running services. With Network Function Virtualization (ETSI NFV), Telecom providers seek to exploit advances in virtualization techniques in order to allow creation of large-scale network services in an agile, seamless and cost-effective manner. As an evolution of this trend, the EU FP7 UNIFY project [1] sets out to integrate modern cloud computing and networking technologies by considering the entire network as a unified service production environment, joining the vast networking assets and data centres of Telecom providers in the same management model. By allowing the agile deployment of new services with seamless instantiation across the entire infrastructure, UNIFY will extend the capabilities of existing NFV technologies.

UNIFY also acknowledges the need for simplified and automated management processes required for future software defined telecommunications environments. In particular, the integration of capabilities that are related to observability, troubleshooting and verification of service components and physical resources is crucial for service providers to improve the cost-efficiency of maintenance, resiliency, and agile deployment of services. Inspired by the DevOps paradigm popular in modern data centers, in UNIFY we refer to these capabilities collectively as *Service Provider DevOps*.

On one hand, orchestration and control layers in software defined "everything" architectures are more than ever in need of capabilities that allow determination of up-to-date, accurate and detailed descriptions of infrastructure utilization and performance of virtual network functions. On the other hand, existing monitoring approaches, such as counter-based monitoring via OpenFlow, impose a centralized client-server communications and thus suffer scalability limitations in the face of highly distributed assets as given in current telecommunications environments.

In this paper, we first introduce the UNIFY Service Provider DevOps (SP-DevOps) concept (Section II). We will then discuss how software defined monitoring, as a key part of SP-DevOps, can be realized by taking advantage of processing capabilities on future programmable infrastructure nodes. In Section III, we specifically address scalability limitations by defining a control-data plane split for monitoring functions (MFs) and introducing a scalable bus for communicating between monitoring function components and other entities in control, orchestration, and management layers. Section IV presents a rate monitoring function as an example of a split MF, and Section V describes the messaging bus in detail. We numerically discuss the scalability gains of software defined monitoring in Section VI, before we conclude the paper in Section VII.

II. SP-DEVOPS CONCEPT

UNIFY aims to reach a high level of agility for service innovation by providing dynamic service programming and orchestration, deploying logical service components (i.e. virtual network functions - VNFs) across multiple network nodes. The UNIFY architecture follows SDN principles with a logically centralized control and orchestration plane. Additionally, compute, storage and network abstractions are combined into a joint programmatic interface referred to as Network Function Forwarding Graph (NF-FG). An NF-FG defines a selected mapping of VNFs and their forwarding overlay definition into the virtualized resources presented by the underlying layer. In order to enable "processing anywhere", another essential goal of UNIFY is the development of a highly programmable network node, called Universal Node (UN). An UN is built on standard COTS (commercial off-the-shelf) hardware platforms (e.g., x86) to provide an execution environment for advanced VNFs, while still meeting carrier-grade network requirements in terms of performance and availability.

To cope with the high service velocity dynamicity enabled by UNIFY, we consider a novel management and operation paradigm for Service Providers, called Service Provider DevOps - *SP-DevOps* [2]. Modern agile software development and operations methods, so far applied primarily in data centre environments (collectively called *DevOps*), constitute a good source of inspiration for future NFV-inspired telecom carrier environments, specifically when rapid and flexible service deployment is targeted as in UNIFY. In SP-DevOps, we follow the same major underlying principles as identified for DevOps [3]: i) *Monitor and validate operational quality*; ii) *Develop and test against production-like systems*; iii) *Deploy with repeatable, reliable processes*; and iv) *Amplify feedback loops*.

Technical aspects associated to these principles reflect on processes and associated tools for monitoring, validating and testing software and programmable infrastructure. While we acknowledge that DevOps has a crucial cultural dimension (reflected barely by *Amplify feedback loops*), our work focuses on technical aspects in terms of tools and infrastructure support.

Even if significant parts of the telecommunication networks are foreseen to be virtualized in the future, we identified important characteristics of telecommunication networks that differ from traditional data centres, i.e.:

- Higher spatial distribution, as telecom resources are spread over wide areas due to coverage requirements
- Lower levels of redundancy in access and aggregation networks compared to the massive data centers of typical cloud computing companies
- Stronger requirements on high availability and latency in according to standards and customer expectations

These differences pose new challenges on DevOps principles applied in telecommunications environments. For instance, operator networks put stricter demands on ongoing verification of service definitions and configurations due to higher failure costs associated with high spatial distribution and lower levels of redundancy. Also scalability of observability is a major issue in logical and highly geographically distributed operator infrastructures (Section III).

SP-DevOps addresses these and further challenges [2], [4] with a set of technical processes supporting developer and operator roles in a virtualized telecom network. Figure 1 illustrates the relation between SP-DevOps processes and the developer/operator roles by means of a service creation lifecycle. The four SP-DevOps processes follow the DevOps principles to meet specific challenges regarding *Observability and Troubleshooting* (Principle: Monitor and validate operation quality); *Verification* (Principle: Deploy with repeatable, reliable processes); and *Development* (Principle: Develop and text against production-like systems).

We also identified three main roles involved in the processes: the *Service Developer*, who assembles the service graph for a particular category of services similarly to the more classical operator role; the *VNF Developer*, who programs virtual network functions and is associated to the classical

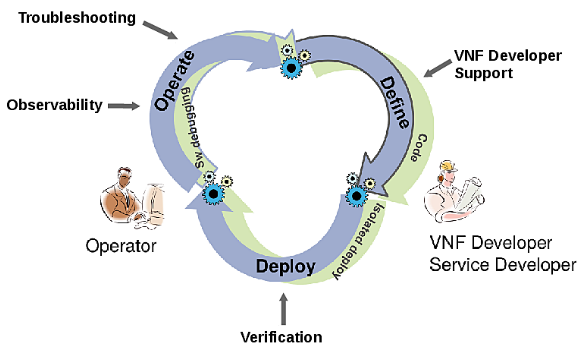

Fig. 1. SP-DevOps cycle for UNIFY service creation

equipment vendor role; and, the *Operator*, whose role is to ensure that a set of performance indicators associated to a service are met when the service is deployed on a virtual infrastructure within the domain of a telecom provider.

One of the main SP-DevOps solutions developed in UNIFY is a reference implementation of an integrated monitoring function (MF). In the following section we will outline this generic concept of software defined monitoring, which adresses scalability issues typically associated with the aforementioned characteristics of telecommunication networks.

III. Software Defined Monitoring

UNIFY proposes an architecture and related interfaces and abstractions to jointly orchestrate and control both network and distributed cloud infrastructure with the promise of increased service agility and flexibility in service deployment. In order to cater for dynamic service management at short time-scales as well as for reliable operations of large-scale networks, the orchestration and control layers are more than ever in need of capabilities that allow determination of up-to-date, accurate and detailed description of the utilization of the infrastructure and the performance of VNFs executing within the environment. However, capabilities for extracting fine-grained monitoring information are currently not sufficiently supported in existing solutions. Our earlier review of SDN and cloud management literature [4], [5][1] revealed a number of open issues for management of such an highly distributed service creation platform. First, the generic approaches of embedding counter-based monitoring capabilities and notifications in control and management protocols such as OpenFlow and SNMP, or container monitoring tools exposing REST APIs, have scalability and resource-efficiency limitations related to frequent and fine-grained observability updates from many distributed nodes. This has negative consequences for time-critical management operations, such as early detection of performance degradations and fast mitigation of network failures. Secondly, we identified the need for programming interfaces that enable tools to exchange rich diagnostic data in a systematic manner with control, orchestration and management components in order to support higher degrees of automation.

In order to meet carrier grade requirements, the need to perform certain management functions in a distributed fashion close to the data-plane has been discussed earlier in [6].

[1] [5] contains a state of the art review of 30+ tools with a focus on SDN and cloud monitoring, troubleshooting, verification, testing and debugging.

The concept of partially decentralized control applies to most fault/performance management tasks in a dynamic and widely distributed environment as service-aware provider networks.

In this paper we present a software defined monitoring solution consisting of two parts:

i) A *Monitoring Function* (MF) reference implementation, realizing software defined monitoring by defining a control-data plane with infrastructure-level *Observability Points* (OPs) (Section IV). Note that the proposed control-data plane split assumes advanced programmability and processing capabilities offered by programmable nodes.
ii) A scalable and flexible messaging bus for communication between MFs, their OPs and other entities in control, orchestration and management layers (Section V).

Our approach extends node capabilities such that certain monitoring, verification and troubleshooting functions can be performed locally at the node - yet still under control of logically centralized control applications. Such MFs are of average complexity - more than a counter, but less complex than big data analytics. This enables high-granular monitoring, aggregation, filtering, and (pre-)processing of measurements (e.g. light-weight statistical analysis), as well as representation of monitoring information in a compact form (e.g. parameter estimates). Such approach can significantly increase the network observability and ensure minimal fault detection and reaction times, while minimizing network and controller load, thereby solving a major scalability problem.

Before going into more detail about an example MF the messaging bus for monitoring data, we will introduce the general concept of our software defined monitoring solution assuming a generic programmable nodes.

Monitoring components

To address the scalability and observability challenges, we introduce the software defined monitoring functions consisting of the following components (Fig. 2):

Monitoring functions (MFs) typically implement functionality for collecting resource (CPU, memory, and storage) and network performance metrics (e.g. link delay, jitter and loss). MFs may not only collect data, but also pre-process monitoring information (e.g. aggregate, filter, etc.) from other MFs across one or several UNs. An MF is implemented as one or several Observability Points (OP) and an MF control app.

An *MF control app* is the software defined monitoring equivalent to SDN control apps, i.e. it is a logically centralized control application taking care of the configuration of OPs and parts of the processing to be performed within the scope of the MF.

An *observability point* (OP) is a MF component that runs locally on the UNs. In general, the implementation of OP capabilities encompasses measurement or verification mechanisms, node-local aggregation and analytics, as well as communication between OPs, depending on the type of the MF. The OP operates in terms of a *local control plane* (LCP) and *local data plane* (LDP).

The *LCP* is splitting certain control functionality from the MF control app for scalability and resource consumption

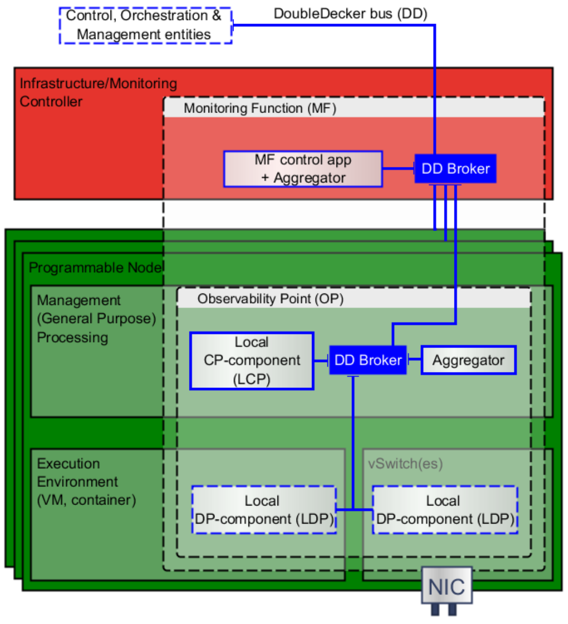

Fig. 2. Overview of the components for local monitoring in programmable nodes. The DoubleDecker bus and its brokers are depicted in solid blue, DoubleDecker clients have blue frames.

purposes. It reflects essentially a local monitoring controller which provides functions for retrieving data from the LDP; processes obtained data; and controls the monitoring behavior (e.g. measurement intensity).

The *LDP* is basically any kind of data source in the infrastructure node (e.g. statistics from logical switches, resource metrics from the VNF executing environment, hardware counters, meters, injected packets, log data, etc.), retrievable from the virtualized environments, the OS or the hardware.

A *messaging bus* enables information exchange between LCPs within one UN and between LCP and their MF control apps, as well as between MFs, and with other entities in control, orchestration and management layers (logging DB, analytics engine, etc.).

IV. In-Network Monitoring and Troubleshooting with Local Control Plane Components

In this section, we detail an implemented example of a monitoring function following the software defined monitoring constructs introduced in section III. The link utilization and rate MF is able to implement a scalable congestion detector [7] based on the analysis of the rate distribution on individual links at several time scales. For this purpose, we employ a statistical method for node-local analytics based on the use of two byte counters for storing the first and second statistical moment ($s_1 = \Sigma x_i/n, s_1 = \Sigma x_i^2/n$) where each x_i represents either the size of a single packet, or another high rate update of the byte throughput. Assuming a log-normal distribution for the observed rates, the parameters of the distribution can be estimated from the statistical moments:

$$\begin{cases} \hat{\mu} = \ln M - \frac{1}{2}\hat{\sigma}^2 \\ \hat{\sigma}^2 = \ln\left(1 + \frac{V}{M^2}\right) \end{cases} \tag{1}$$

where M and V are the sample mean and variance. Once the estimates are obtained, the cumulative density function (CDF) can be used for detecting increased risks of link overload by inspecting the percentiles.

Existing approaches (e.g. SNMP [8] and sFlow [9]) normally involve forwarding of raw measurement information to dedicated monitoring equipment for further processing, which impacts the scale at which monitoring can be efficiently performed and thereby the overall network observability. For this reason, standard practice for identifying increased bandwidth consumption is based on low-frequency counter inspections and reporting when the average exceeds a fixed threshold. Using such low-resolution averages often leads to missed congestion episodes as well as false alarms, as these averages usually are far below the link capacity and determination of suitable detection thresholds is difficult.

In contrast, local querying of counter statistics (i.e. in an LCP) at high rates enables high accuracy in the captured aspects of the traffic behaviour, with significantly lower overhead compared to processing raw measurements at a dedicated monitoring station. By varying the querying frequency, high-granularity monitoring information can be provided to management layers in a flexible and scalable manner and in a compact form as parameter estimates, without the cost of constant high rate sampling of the counters. Reporting of parameter estimates enables efficient dissemination of rich statistical information about the observed traffic rate behavior that can be further used for analytic or predictive purposes. By inspecting the CDF of the obtained parameter estimates the risk of congestion can be predicted at varying time scales. For efficient troubleshooting, the predicted risk is a significantly more robust indicator of persistent congestion than the average traffic rate.

The approach has been initially evaluated [7] with respect to estimation error and congestion detection rate at varying time scales in a stand-alone simulator framework implemented in Scala, using data from 1Gb/s and 10Gb/s links. Using a naive congestion detector based on the CDF, yields a success rate of over 98% when detecting episodes of high congestion risk at 0.3 s using estimates captured at 5 m intervals.

Implementation on programmable nodes

The rate monitoring MF control app operates on a dedicated (SDN) controller while the OP (LCP+LDP) operates locally in a network node. The control app forwards incoming monitoring information and configuration requests to the active OP. Resulting monitoring information is pushed on the DoubleDecker bus (Section V) to directly notify a receiver (e.g. a elastic network function), to publish on the sub/pub interface, or to be stored in a (distributed) monitoring or logging DB.

A rate monitoring MF may consist of one or several OPs depending on the monitoring needs. For one OP, the node-local statistical modeling takes place in the LCP (see Figure 2) and is based on the reading of two counters in the LDP, for storing the first and second statistical moments. This deviates from the current practice of using a single counter for each of the byte and packet rates, and hence the available set of counters in virtual switches needs to be extended with corresponding sum-of-squares counters. The only additional high rate operation required to compute and store the second moment is one multiplication, one addition and one write to a (long) integer register. An additional counter slightly increases the LDP overhead, but enables accurate and light-weight node local modeling at lower rates via method-of-moments estimates, which can also be reported to the management layers in a scalable and compact form. The LCP also implements functions for reading LDP counters. Depending on the scope of monitoring, this can be done in different ways: for example, if the LDP is an OpenFlow (OF) switch, the LCP acts as a mini controller using the OF protocol to retrieve OF counters; alternatively, for monitoring physical NICs, file access (such as /proc in the Linux operating system) or access to shared memory can be used to access counters by an LCP process.

The rate monitoring tool supports the SP-DevOps concept by its technical design, which offers a scalable approach to increased network observability based on node-local modeling of counter data with low computational complexity. The statistical model that we use can be applied for autonomous service and network management, including troubleshooting support and dynamic resource management.

V. DOUBLEDECKER: A FLEXIBLE AND SCALABLE MESSAGING BUS

As mentioned, there are numerous challenges for efficient monitoring in the context of SDN. A way to reach a good observability while limiting performance impairments is the distribution of data treatment close to the measurement points, so that data could be aggregated and processed without imposing all the overhead on a central controller.

The DoubleDecker messaging bus has been designed to provide a scalable system for easily integrating different monitoring functions, and transporting their results. It is routing messages between connected applications, allowing point-to-point synchronous and asynchronous communication, as well as providing a lightweight publish/subscribe interface.

A main problem the messaging bus is trying to solve is to provide a simple and easy to use transportation mechanism for the various monitoring functions. As these monitoring functions can be quite different in their complexity and implementation, the solution should be easy to apply in a range of scenarios (e.g. counters read by an SDN Controller, daemons running on a distributed node, dynamically instantiated VMs, etc.). Additionally, it should be easy to use in dynamic and changing environments as envisioned in UNIFY, where functions may migrate and scale-in/out depending on various factors such as incoming traffic load. Finally, it should be easy to implement support for the messaging system in the clients, and to integrate with existing systems. For these reasons we have chosen to implement the DoubleDecker messaging bus as a protocol running on top of the ZeroMQ messaging library. ZeroMQ implements the ZeroMQ Message Transport Protocol in a C library with bindings and native implementations available for many languages (currently more than 20). While ZeroMQ does not support many features of more complex messaging system out of the box, it can be tailored for many scenarios. ZeroMQ is also attractive since it is designed with scalable and low-latency messaging in mind. The core of ZeroMQ is a set of socket abstractions, similar to traditional BSD sockets for UDP and TCP, which provide the building blocks for creating various messaging patterns.

The bus consists of two components, brokers and clients, both implemented on top of the ZeroMQ library. The brokers are responsible for routing messages between the clients, and can be connected to each other in a hierarchical, tree-like, fashion. Messages between clients connected to different brokers will take the shortest path through the tree structure, off-loading more central brokers by keeping messages as local as possible. A client connects to the closest broker (e.g. a broker running on the same machine) using a simple protocol and one of the supported transport protocols. In a data center thousands of clients can be deployed without overloading the link toward a central controller. They can join or leave the bus dynamically meeting scalability and flexibility requirements.

A client registers itself with a name which will be its unique identifier. Identifiers are independent of the network address so clients can be moved completely transparently for other clients, with no other configuration needed. Dynamic migration and scaling of VNFs as envisioned by UNIFY is thus supported very effectively by the DoubleDecker bus, which absorbs much of the complexity. The initial configuration is also simplified as locally unique IP addresses can be used. Keeping messages local is recommended for efficiency but this is not a limitation. Local nodes can also communicate with higher layer components which in a SDN network can be physically far away.

The local access to the bus allows partitioning robustness. If a part of the bus is isolated, messaging between two clients connected locally to the same broker will work transparently. The broker will survive and periodically try to reconnect with the higher level. Even a local failure disconnecting the client to its broker is not too damaging, the clients will have the same behavior as the brokers and try to reconnect. Temporary caching can be used to prevent message loss.

The clients have minimal knowledge of the actual architecture and are fitted for specific tasks. The clients have to follow a standardized protocol to communicated with the bus and have to define their own sub-protocol to communicate with other clients. The main protocol defines the way clients and brokers communicate. Clients must implement the functions allowing the identification in the bus as well as the heartbeat mechanism. The client has to check periodically if the broker is still running. Inversely, the broker will not initiate any messaging but will periodically delete the clients who have been quiet for too long. Apart from this the client can extend the protocol to communicate with other clients with very specialized tasks. A lot of work has been put into keeping this integration as simple as possible. As the bus has a tree-like topology, inter-broker communication is required. The main task of the broker is to route messages through the architecture. In the case of two local clients the forwarding is trivial. Otherwise the broker forwards the message to the upper broker. Brokers implement the same heartbeat mechanism as between clients and brokers.

When connected, the clients can use the four simple functions listed in the Table I to communicate with each other. Any other functionality is up to the developer or user to implement. For instance in the case of the rate monitoring function (Section IV), the rate monitoring algorithm has been embedded in a client able to transmitting the calculated values to other components.

TABLE I. EXAMPLE OF DOUBLEDECKER CLIENT FUNCTIONS

Function	Purpose
send(id, data)	Client-to-Client transmission of data to Client *id*.
[id, data] = receive()	Receive data from name, Client-to-Client.
subscribe(topic)	Subscribe to messages beginning with prefix *topic*. A subscription for topic A receives e.g. messages published both on topics A and ABCD.
publish(topic, data)	Publish message in the topic with prefix *topic*.

In addition to the basic functionality, we have implemented a publish/subscribe mechanism. Clients can subscribe to publication lists so they receive all the messages sent on this list. Groups can be formed according to the usage of the client or the physical location for instance. The messaging capabilities remain the same as for the peer to peer communication. Again the brokers hide most of the implementation challenges to keep the client simple to use and extend.

With the client side of the protocol being relatively simple, it is easy to implement in any of the supported languages. It is also easy to extend the client modules to act as a proxy to existing tools, e.g. integration with a REST API can be done in just a few lines of code. Implementations in Python3, C and Java exist to show the modularity of the bus and demonstrate the reference implementation.

In an earlier demonstration [10], we used the Python3 version to connect MFs with a load-balancer SDN app and a database. For increased performance, we have since implemented and tested the performances on a C version. With a simple set-up we can reach 1Gb/s (limit of the link used). To reach this rate we send messages as big as 1MB to minimize the time spend resolving the route and maximize the throughput. If we instead send small messages, we can currently reach 130k messages per second. Even in the case of a very accurate monitoring e.g values sent every ms, the bus should not be the bottleneck.

We have also shown that DoubleDecker can be integrated inside Docker containers [11]. Docker allows sharing directories between containers, making it possible to share files for inter process communication, removing the need for IP addressing. Containers can be linked together to share environment variables or expose ports to each other. Embedding the components of the bus in containers allows a better portability and security by isolating them from the host machine. If the containers are moved, the same properties apply, no extra configuration will be needed in other clients messaging the container VNF.

VI. DISCUSSION

A functional verification of our software defined monitoring concept has been demonstrated in [10]. In this demo, earlier versions of the messaging bus and rate monitoring function have been combined with a rudimentary configuration tool and an OpenFlow control plane verification mechanism. Via the DoubleDecker bus, link rate data was efficiently logged in a time-series database. Detected link congestions automatically triggered a load balancing mechanism.

To provide an initial assessment of the scalability gains by software defined monitoring (realized as a combination of rate monitoring and DoubleDecker messaging), we developed a simple model to compare the amount of transmitted messages

TABLE II. MESSAGE/DATA RATE COMPARISON OF RATEMON AND DOUBLEDECKER (DD) WITH SNMP AND OPENFLOW (OF) BY SAVING FACTORS

Solution	Read Rate (/sec)	Message Rate (/sec)	Saving SNMP	Saving OF	Bandwidth (Mb/s)	Saving SNMP	Saving OF
SNMP	1000.0	822600000			627594		
OpenFlow Multipart	1000.0	193054000			602465		
RateMon	0.3	246780	$3 * 10^3$	$0.7 * 10^3$	154	$4 * 10^3$	$3.9 * 10^3$
RateMon + DD server-level aggr.	1.0	51413	$16 * 10^3$	$3.7 * 10^3$	49	$13 * 10^3$	$12.3 * 10^3$
RateMon + DD rack-level aggr.	1.0	1072	$770 * 10^3$	$180.0 * 10^3$	49	$13 * 10^3$	$12.3 * 10^3$

and data with respect to a standard SNMP-based option and an standard OpenFlow-based option. As a reference network, we take a data center with 1000 server racks, 48 servers per rack, 4 virtual switches executed within each server and 4 ports for each of the virtual switches. The network fabric is a classic three-tier architecture, with Top-Of-the-Rack switches of 52 ports (48 downlinks and 4 uplinks), 48-port switches in the second tier and 100-port switches in the third tier. We assume that all network ports from both virtual and physical switches are being monitored. To estimate the message overhead, we account only for VXLAN encapsulation along with IPv4, UDP headers. The SNMPv3 header with no SNMP context and 1 byte for message security parameters is counted as part of the overhead of the SNMP solution. For OpenFlow, we considered the OpenFlow v1.5 header with multipart replies and the port statistics replies to a request that uses the OF_ANY option to aggregate the counters from all the ports of a switch in one message. All counters transmitted are considered to be 64-bit long in order to support high-speed interconnects. The data read and message rates were chosen in line with the RateMon results from [7].

For the sake of simplicity, we assume that DoubleDecker implements an addressing and context scheme identical to the SNMP Scoped PDU, and thus disregard this part of the message from the overhead calculation. Table II presents the results of the estimate, and shows that a naive SNMP implementation would generate 4000x - 13000x more data onto the network to provide essentially the same information as the combination of Rate Monitoring and DoubleDecker. Compared with OpenFlow with multipart aggregation, our proposal provides similar large improvements in terms of bandwidth usage. The aggregation within DoubleDecker reduces the message rate by a factor of 5x to 230x compared to the direct dissemination of Rate Monitoring results, depending on where the aggregation is performed, even though we increased the results read rate by 3x for timeliness purposes. One server connected on a Gigabit Ethernet connection could thus very comfortably handle the amount of data generated by the Rate Monitoring and transmitted with DoubleDecker, with significant network and compute capacity left to perform other tasks. In addition to capacity for handling a very high message rate, both the SNMP and OpenFlow-based solutions would require significant compute resources to calculate the rate estimate from massive amounts of raw data.

VII. CONCLUSIONS

The UNIFY project integrated modern cloud and networking technologies by considering the entire infrastructure as one unified service production environment. This will offer telecom providers a fully virtualized, programmable service creation platform enabling fast and dynamic service innovation. Besides a unified service orchestration and programming architecture, UNIFY focused on SP-DevOps, a novel management paradigm designed to cope with the increased service dynamicity. One challenge is to provide up-to-date, accurate, and detailed monitoring information to orchestration and control layers in a scalable way. We proposed a software defined monitoring approach, defined a control plane split for monitoring functions, and introduced a scalable bus for communicating between MFs and higher control layers. We used the example of a link rate monitoring function to show how the processing capabilities provided by programmable network infrastructure allow delegation of selected MF functionality to local control plane components. Software defined monitoring with our messaging bus enables node-local aggregation and pre-processing of measurement results, providing telecom operators with a way to handle the tradeoff between high-granular observability and network/compute resources load at central control/management layers. Our numerical analysis shows that our software defined rate monitoring generates only a tiny fraction of the monitoring traffic from comparable SNMP and OpenFlow implementations, while providing the same information granularity.

ACKNOWLEDGMENT

The authors would like to thank Fulvio Risso and Juhoon Kim for valuable discussions. The research leading to these results has received funding from the EU Seventh Framework Programme under grant agreement nr. 619609 (UNIFY).

REFERENCES

[1] A. Csaszar, W. John, M. Kind, C. Meirosu, G. Pongracz, D. Staessens, A. Takacs, and F.-J. Westphal, "Unifying cloud and carrier network: Eu fp7 project unify," in *IEEE/ACM UCC'13*, 2013.

[2] J. Kim, C. Meirosu, I. Papafili, R. Steinert, S. Sharma, F.-J. Westphal, M. Kind, A. Shukla, F. Nemeth, and A. Manzalini, "Service provider devops for large scale modern network services," in *IFIP/IEEE IM'15, BDIM Workshop*, 2015.

[3] S. Sharma and B. Coyne, *"DevOps for Dummies"*. IBM limited edition, 2013.

[4] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu, "Research directions in network service chaining," in *IEEE SDN4FNS'13*, 2013.

[5] W. John and C. Meirosu, "Unify d4.1: Initial requirements for the sp-devops concept, universal node capabilities and proposed tools," 2014, online: https://www.fp7-unify.eu/index.php/results.html#Deliverables.

[6] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Skoldstrom, "Scalable fault management for openflow," in *IEEE ICC*, 2012.

[7] P. Kreuger and R. Steinert, "Scalable in-network rate monitoring," in *IFIP/IEEE IM'15)*, 2015.

[8] R. Presuhn, "Management information base (MIB) for the simple network management protocol (SNMP)," 2002, RFC 3418, Internet Engineering Task Force.

[9] P. Phaal, S. Panchen, and N. McKee, "Inmon corporations sflow: A method for monitoring traffic in switched and routed networks," RFC 3176, Tech. Rep., 2001.

[10] F. Nemeth, R. Steinert, P. Kreuger, and P. Skoldstrom, "Roles of devops tools in an automated, dynamic service creation architecture," in *IFIP/IEEE IM'15, Demo Session*, 2015.

[11] F. Moradi, B. Pechenot, and J. Martensson, "Monitoring transport and cloud for network functions virtualization," in *EWSDN'15 demo session*, 2015, video online: https://vimeo.com/130979415.