RESCOM 2017 Summer school

Damien Saucez, Adrien Lebre, Stefano Secci

RESCOM 2017

Summer school

VIRTUALIZATION

June 19-23, 2017
Le Croisic, France

Main Courses:
Ubiquitous Computing: from Mainframe to Clouds (Adrien Lebre, Inria)
Introduction to Software Defined Networks (Giuseppe Bianchi, Roma III)
Mobile Edge and Fog Computing (Guillaume Pierre, IRISA)
Virtual Machines placement (Fabien Hermenier, I3S)
Network Function Virtualisation orchestration (Vania Conan, Thales)
Online and Global Network Optimization with SDN (Jérémie Leguay, Huawei)
Big Data Processing in the Cloud: Hadoop and Beyond (Shadi Ibrahim, Inria)
Edge Computing for IoT Application Scenarios (Paolo Bellavista, U. Bologna)
Scaling the experiments (Lucas Nussbaum, LORIA)

Numerous labs:
OpenStack, SDN, Cloud Orchestration, ...

more information at http://rescom2017.univ-nantes.fr

Preface

This volume contains the extended abstract of the posters presented at RESCOM'17 Summer School. RESCOM 2017 held on June 19-23, 2017 in Le Croisic.

There were 14 submissions. The committee decided to accept 11 extended abstracts.

The RESCOM school is a key event of the French scientific landscape that gathers academics as well as industrials experts during one week to present and discuss about the latest challenges in Information and Communications (ICT) technologies.

Building on previous successful and highly attended editions of the Cloud Days organized by the transversal action "Virtualization and Clouds" of the RSD GDR, the 2017 RESCOM edition aims at further bridging the gap between the communication networks and distributed systems experts of the RSD GDR. Although the use of virtualization technologies has been leading to exciting developments for almost ten years now, current solutions for operating virtual infrastructures still need to provide a tighter integration between network, computation and storage resources at a geographically wider area network scope. This integration is mandatory to build the next generation of internet backbones (SDN, NFV...) and to cope with the new requirements of IoT and mobile computing applications that cannot be satisfied by current cloud computing offers.

To favor this cloud/network technology convergence, along the lines of the GDR Cloud Days of gathering together people with a diverse set of backgrounds, we built the 2017 program in order to define common bases and highlight current challenges addressed by the network and distributed system communities. We will discuss in particular how virtualization technologies are used in various contexts (mobile, infrastructure, data-centers, edge networks) and present major challenges in the research agenda. The program shows a balanced mix of general lectures and practical sessions with the goal of giving to attendees an large overview while being able to discover and deal with technical aspects.

We would like to thanks all the authors for proposing their posters and thanks the sponsors for their support.

We appreciate the support given by EasyChair to manage submissions.

June 19, 2017
Le Croisic

Table of Contents

Efficient Container Deployment in Edge Computing Platforms

Arif Ahmed and Guillaume Pierre
IRISA / University of Rennes 1

I. Introduction

Cloud computing data centers are composed of powerful computing nodes connected by reliable backbone networks. However, these resources are concentrated in a small number of data centers. The latency between an end user and the closest available cloud data center comes in the range of 20-40 ms (over wired networks) and up to 150 ms (over 4G mobile networks). A number of latency-sensitive applications (e.g., augmented reality) require extremely low end-to-end latencies and therefore cannot make use of traditional cloud platforms.

An obvious solution to address this problem is to place cloud server nodes extremely close to the users, within a couple of network hops. In Edge Computing, computational nodes are broadly distributed in a large number of geographical locations so computation capacity is always available in immediate proximity of any end user. As an added benefit, deploying server nodes in many locations may also increase the availability of the application.

The required number of nodes to cover a sufficiently large area suggests that each node will have to be small and cheap, while the broad distribution of resources indicates that their respective connection to the Internet will use relatively slow commodity networks. However, deploying cloud applications in such conditions can unfortunately be painfully slow. The goal of this work is therefore to understand the source of inefficiencies in this deployment process, and to propose optimizations targeted toward edge computing platforms composed of a very large number of weak nodes connected to the Internet via commodity network links.

II. State of the art

Container deployment in fog computing platforms is being studied increasingly. For example, Bellavista et al studied the feasibility of docker-based container deployment in fog computing for IoT-based applications [1]. They concluded that with proper configuration and optimization, scalability and low overhead can be achieved in fog nodes based on Raspberry Pis. However, they measured deployment times without proposing techniques to speed up this container deployment.

Harter et al proposed an optimized docker storage driver to speed up container deployment [2]. Using deduplication between Docker images as well as lazy loading, the proposed driver significantly speeds up the deployment of standard Docker images. However, they performed their study in traditional cluster-based environment which is quite different from edge computing platform. We aim to produce similar work specifically targeted toward edge computing environments.

III. Experimental setup

To study application deployment times in edge computing environments we first deployed a small experimental testbed [3]. The testbed has 3 worker nodes and one master to form a small edge cloud.

A. Raspberry Pi

Raspberry Pi machines have become a promising enabler for edge computing platforms: they are cheap, small, easily portable, energy-efficient, and they can be connected with other sensor devices. Moreover we found that, despite their many hardware limitations, they are still powerful enough to run many serious cloud applications [4]. In our testbed, Raspberry PIs use their integrated Wi-Fi interface to communicate with nearby clients, and their wired Ethernet device to connect to the Internet.

Our testbed currently comprises three "Raspberry PI 3" machines acting as worker nodes, and one HP Laptop (16 GB RAM, Intel Core i7 Processor) acting as the master node which coordinates the whole setup.

B. Container-based virtualization

Raspberry PIs are based on ARM v9 processors, which do have support for hardware-based virtualization. However, we found that container-based virtualization is much more leightweight and suitable for small machines such as Raspberry PIs. Container virtualization is a technique that enable to run multiple applications sharing same underlying OS kernel. The image size of a container is very small (because it does not need to include an operating system kernel). The above features compel to use containers for cloud deployment on edge platforms rather than traditional VM-based cloud.

C. Docker and Kubernetes

Docker is a popular open source container deployment tool with strong community support. It makes it very easy to package software and it is compatible with many different architectures. This makes it a very useful tool to deploy software on an edge computing platform. To deploy applications over a cluster of geographically-distributed worker nodes we rely on Kubernetes. Kubernetes executes over Docker and supplements it with extra suppoprt for application deployment, maintenance, horizontal scaling, etc.

In our testbed the Kubernetes master which is responsible for deploying containers is running in the master node; other worker nodes run only the Kubernetes worker. We used Docker version 1.4 and Kubernetes version 1.6 for the cloud orchestration.

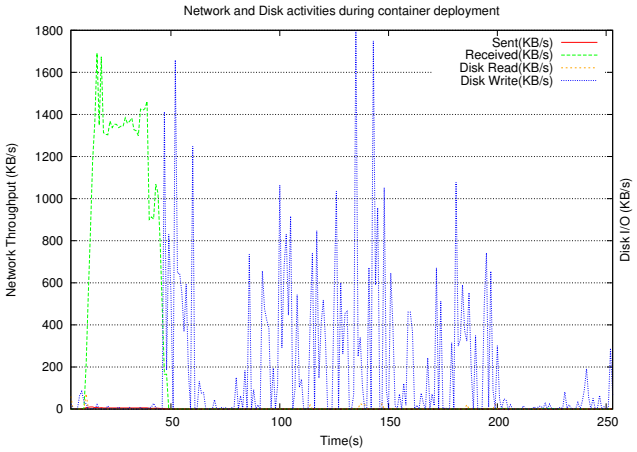


Fig. 1. Resource usage during a Docker container deployment

IV. Deployment time analysis

Our goal during the experimentation is to find the different behaviours that impact the container deployment on edge computing platforms.

The experimentation was carried out deploying the popular "Ubuntu" docker image. Container deployment in Docker may happen in two ways: firstly, the image is already available in the worker node, and it can be started immediately; secondly, the image is not available, and this requires Docker to pull the image from the centralised docker repository. Throughout the experimentation, we assume that the docker image is not available locally and therefore needs to be pulled from the repository. During the container deployment we measure the total deployment time as well as the instantaneous network and disk bandwidth to better understand the detailed container deployment process. We used the standard `nethogs` tool to capture the bandwidth activity, and `iostat` to trace the read and write throughput of the disk at 1-second granularity.

Figure 1 shows the bandwidth and network activities during the deployment on Ubuntu container. The total time to deploy Ubuntu container is about 253 seconds. After giving the container deployment command, the bandwidth activities start at the worker node indicating the docker had started pulling the image layers. During the pull operation (which lasts about 45 seconds), there is no disk I/O activity whatsoever. After the download phase, the system starts extracting the image. When the container image is composed of multiple layers, Docker downloads all layers simultaneously, then extracts the respective layers sequentially.

Although this organization is very sensible for powerful and well-connected server machines, it includes inefficiencies in our edge computing scenario. First, downloading multiple files simultaneously across a slow network connection unnecessarily delays the moment when any of these downloads completes at the client side. Second, we see in Figure 1 that only one of the I/O devices is fully utilized at any moment in time (either the network or the disk), but never both simultaneously.

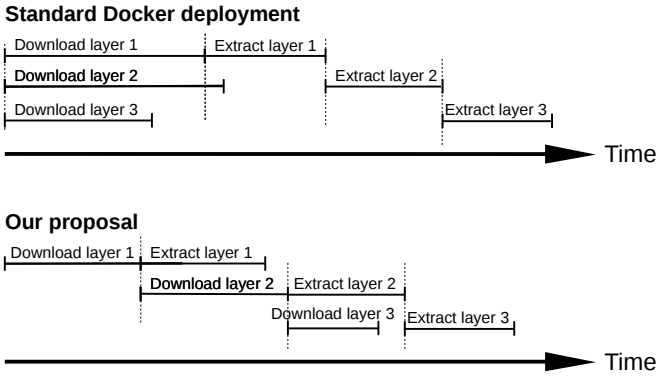


Fig. 2. Standard and proposed container deployment processes

V. Deployment time optimization

Figure 2 shows on top the standard Docker container deployment process: image layers are first downloaded in parallel, then extracted sequentially to the disk. Since multiple layers are downloaded in parallel, there is a network bottleneck which slows down the deployment process.

We are currently modifying the standard Docker pull operation to optimize the deployment time. As shown at the bottom of Figure 2, the proposed model pulls the Docker image layers sequentially in order to avoid the network bottleneck. It also starts extracting layers as early as possible: we can start extracting layer N as soon as it has been fully downloaded, and layer $N-1$ has finished being extracted.

This part of the work is still ongoing, and we hope to have good results in time to be displayed at the RESCOM summer school. In future work, we will implement this proposed model and compare the performance with existing standard model.

VI. Conclusion

In this work, we have experimentally studied container deployment performance on real edge computing environment. During the deployment time, we traced different metrics:- to study the behaviour of the process. We concluded by proposing a new container deployment model for docker tool that will eliminate the network bottleneck and optimize the overall container deployment time. Although the proposed techniques were designed for weak edge cloud scenarios, we hope they may also more beneficial in more powerful cluster-based environments.

References

[1] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on Docker containerization over RaspberryPi," in *Proc. ICDCN*, 2017.

[2] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Slacker: Fast distribution with lazy Docker containers," in *Proc. Usenix FAST*, 2016.

[3] "The mobile edge cloud testbed at IRISA Myriads team," YouTube video, Jan. 2017, https://www.youtube.com/watch?v=7uLkLitiSPo.

[4] A. van Kempen, T. Crivăţ, B. Trubert, D. Roy, and G. Pierre, "MEC-ConPaaS: An experimental single-board based mobile edge cloud," in *Proc. IEEE Mobile Cloud Conference*, 2017.

Software-Defined Networking: A Self-Adaptive Consistency Model for Distributed SDN Controllers

Fetia Bannour, Sami Souihi and Abdelhamid Mellouk
LISSI/TincNetwork Research Team
University Paris-Est Créteil (UPEC), France
(fetia.bannour, sami.souihi, mellouk)@u-pec.fr

Software-Defined Networking [1] promises to solve the emerging problems facing current networks. Contrary to the decentralized control logic underpinning the devising of the Internet as a complex bundle of box-centric protocols and vertically-integrated solutions, the SDN paradigm advocates the separation of the control logic from hardware and its centralization in software-based controllers. These key principles offer outstanding opportunities to innovate network applications and incorporate automatic and adaptive control aspects, thereby facilitating network management and guaranteeing the users Quality of Experience. Despite all the hype, SDN adoption raises many challenges including the scalability and reliability issues of centralized designs that can be addressed with the physical decentralization of the SDN control plane [2]. However, such physically distributed, but logically centralized systems, bring an additional set of challenges regarding the best approach to designing and decentralizing the SDN control plane while maintaining the logically centralized network view.

In particular, maintaining a consistent and up-to-date global network view for SDN applications running on top of distributed SDN controllers while preserving good performance is a challenging task that should be studied carefully. More specifically, ensuring a proper network state distribution in the SDN control plane requires investigating the required consistency model [3]. Current implementations of distributed SDN controllers use static mono-level [4] or multi-level [5] consistency models such as the strong, eventual and weak state consistency levels. Strong consistency models are usually implemented for applications that are intolerant of state inconsistencies. They guarantee that all controller replicas in an SDN cluster have the most updated network information, but at the cost of increasing communication and synchronization overhead. On the other hand, eventual consistency models are used for developing inconsistency-tolerant applications that require high-availability and scalability. Their limitations are related to their reliance on additional application logic or controller mechanisms for resolving state conflicts and inconsistencies.

For instance, the ONOS controller platform [5] offers different state dissemination mechanisms [6] to achieve a consistent network state across the distributed cluster of ONOS controllers. More specifically, ONOSs distributed core eases the state management and cluster coordination tasks for application developers by providing them with an available set of core building blocks for dealing with different types of distributed control plane state, including a ConsistentMap primitive for state requiring strong consistency and an EventuallyConsistentMap for state tolerating weak consistency.

That way, applications that favor performance over consistency store their state in the shared eventually-consistent data structure that uses optimistic replication assisted by the gossip-based Anti-Entropy protocol [7]. For example, the global network topology state which should be accessible to applications with minimal delays is managed by the Network Topology store according to this eventual consistency model. Recent releases of ONOS treat the network topology view as an in-memory state machine graph. The latter is built and updated in each SDN controller by applying local topology events and replicating them to other controller instances in the cluster in an order-aware fashion based on the events logical timestamps. Potential conflicts and loss of updates due to failure scenarios are resolved by the anti-entropy approach [7] where each controller periodically compares its topology view with that of another randomly-selected controller in order to reconcile possible differences and recover from stale information. On the other hand, state imposing strong consistency guarantees is managed by the second data structure primitive built using RAFT [8], a protocol that achieves consensus via an elected leader controller in charge of replicating the received log updates to follower controllers and then committing these updates upon receipt of confirmation from the majority. The mapping between controllers and switches which is handled by ONOSs Mastership store is an example of a network state that is maintained in a strongly consistent manner.

Accordingly, designing a distributed SDN control plane where different types of network state are statically assigned to various levels of state consistency, in large-scale and dynamic networking environments, does not guarantee real-time trade-offs between application correctness and network performance. Instead, we propose a self-adaptive multi-level consistency model that adapts to an appropriate level of controller consistency based on data criticality according to changing networking conditions. Such a self-adaptive model spares application developers the tedious task of implementing multiple application-specific consistency models. Besides, an adaptively consistent SDN controller platform that tunes in real-time the controller consistency to a suitable level based on specific network requirements is very likely to reduce controller state distribution overhead, ensure network scalability, and enhance the overall performance of SDN applications.

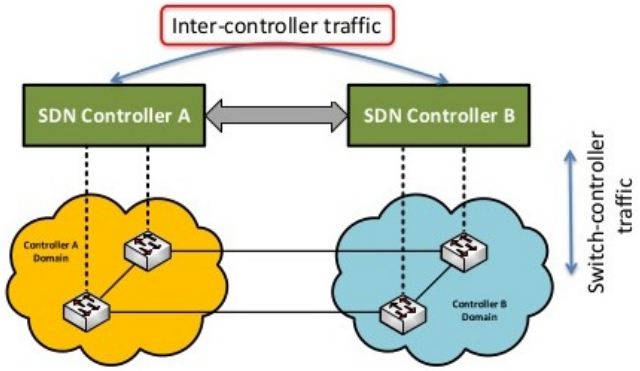


Fig. 1: Distributed SDN Controllers
https://www.slideshare.net/paolo_giaccone/the-role-of-intercontroller-traffic-in-sdn-controllers-placement

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, Software-Defined Networking: A Comprehensive Survey, Proceedings of the IEEE, vol. 103, no. 1, p. 63, 2015.

[2] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, Logically centralized?: state distribution trade-offs in software defined networks, in Proceedings of the first workshop on Hot topics in software defined networks, ser. HotSDN 12, 2012.

[3] F. Botelho, T. A. Ribeiro, P. Ferreira, F. M. V. Ramos and A. Bessani, Design and implementation of a consistent data store for a distributed SDN control plane, 2016 12th European Dependable Computing Conference (EDCC), vol. 00, pp. 169180, 2016.

[4] OpenDayLight Project. [Online]. Available: http://www.opendaylight.org/

[5] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. OConnor, P. Radoslavov, W. Snow, and G. Parulkar, Onos: Towards an open, distributed SDN os, in Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, ser. HotSDN 14. New York, NY, USA: ACM, 2014, pp. 16.

[6] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, Inter-controller traffic in onos clusters for sdn networks, in 2016 IEEE International Conference on Communications (ICC), May 2016, pp. 16.

[7] A. Bianco, P. Giaccone, S. D. Domenico, and T. Zhang, The role of inter-controller traffic for placement of distributed SDN controllers, CoRR, vol. Abs/1605.09268, 2016.

[8] D. Ongaro and J. Ousterhout, In search of an understandable consensus algorithm, in Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference, ser.USENIX ATC14. Berkeley, CA, USA:USENIX Association, 2014, pp. 305320.

A Round-Robin Scheduling for Computer Clusters with Compatibility Constraints

Thomas Bonald*
*Télécom ParisTech
Université Paris-Saclay, France

Céline Comte†*
†Nokia Bell Labs, France

***Abstract*—We consider a computer cluster with some arbitrary bipartite graph of compatibilities between jobs and computers. Assuming jobs can be processed in parallel on several machines, we design a scheduling algorithm achieving balanced fair sharing of the computing resources, similar to round-robin scheduling in the presence of a single computer. We analyze the performance of this algorithm by introducing a new class of queueing system. In this model, the service interruptions and resumptions of the algorithm are reinterpreted in terms of random routing.**

A full version of this paper was submitted to Performance Evaluation Journal. It is available on arXiv [1].

***Index Terms*—Scheduling, parallel processing, order-independent queues, balanced fairness, insensitivity.**

I. Computer cluster

We consider a cluster of computers which can process jobs in parallel. Each computer has a fixed capacity expressed in floating-point operations per second. Each job consists of some random number of floating-point operations called the job size.

Compatibility constraints. When a job enters the cluster, it is assigned to a set of computers. This set may be chosen by the service provider (e.g. to randomly balance the load between machines) or imposed by technical constraints like data availability. We assume that the assignment is independent of the current state of the cluster, and in particular of the number of jobs at each computer.

The set of machines to which a job is assigned defines its class. The possible assignments are then represented by a bipartite *graph of compatibilites* between classes and computers, where there is an edge between a class and a computer if this computer can process the jobs of this class. A toy example is given in Fig. 1.

Parallel processing. Each computer processes its jobs sequentially in first-come first-served (FCFS) order. When a job is in service on several computers, these are pooled so that the service rate of the job is the sum of the capacities of the computers. The validity of this last assumption will be discussed in the next section.

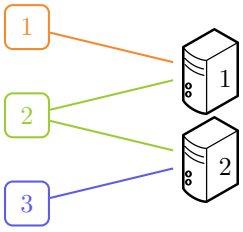


Fig. 1. Bipartite graph of compatibilities. The cluster contains two computers. There are three job classes, one for each possible assignment: class-1 and 3 jobs can only be served by computers 1 and 2, respectively, while class-2 jobs can be served by both computers.

II. Scheduling algorithm

A single computer. Round-robin scheduling consists in allocating equal-size time slices to the jobs in circular order. Equivalently, each job is interrupted once some fixed quantity of floating-point operations was processed.

The system state can be described as an ordered sequence of jobs. The computer processes the job which is at the head. When the service of a job is interrupted, this job is moved to the end of the queue while the computer starts processing the next job. An incoming job is appended to the end of the queue.

Multi-computer extension. Our extension consists in allocating the same quantity of work (expressed in floating-point operations for instance) to each job before service interruption, independently of the number and capacity of the computers that are processing this job.

We propose a distributed implementation where each computer is equipped with an exponentially distributed random timer. The expiration rate of the timer is proportional to the service capacity of the computer. When a job is in service on several machines, its service is interrupted whenever the timer of one of its machine

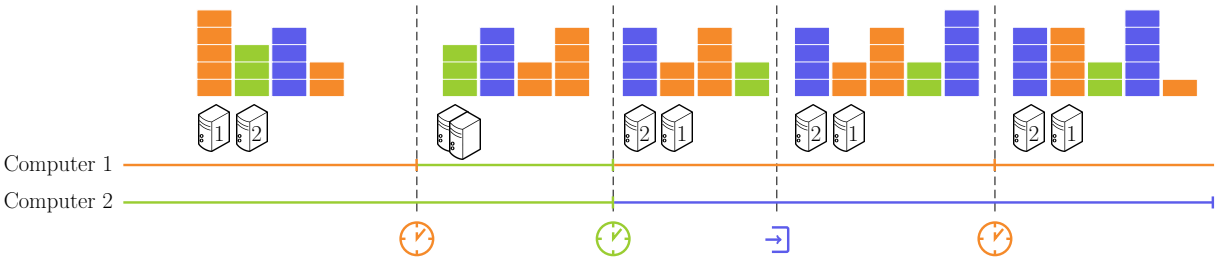


Fig. 2. Example of execution of the scheduling algorithm. The first transition is triggered by the expiration of the timer of computer 1. The first orange job, which was in service on this computer, is moved to the end of the queue. The second transition corresponds to the expiration of one of the two timers, after which the service of the green job is interrupted. Then a blue job arrives and is appended to the end of the queue. Finally, the timer of computer 1 expires and the first orange job is moved to the end of the queue.

expires. The time before interruption is thus exponentially distributed, with a rate which is proportional to the service rate of the job. Hence, the quantity of work given to a job before interruption is on average the same for all jobs. This mean quantity is denoted by θ. Fig. 2 gives an example for the configuration of Fig. 1.

The key parameter is the mean number $m = \frac{\sigma}{\theta}$ of interruptions per job, where σ is the mean job size. From the single-computer case, one can expect that the resource allocation is all the more *insensitive* when m increases, in the sense that the performance only depends on the traffic intensity of each class; however, taking m too large may induce a too high parallelization overhead.

III. Queueing model

Order-independent queue. The system is modeled by an order-independent queue [2] containing as many servers as there are computers in the cluster. Jobs of each class enter as an independent Poisson process. Each server processes its jobs sequentially in FCFS order and each job can be processed in parallel by several servers. Jobs have independent exponentially distributed unit sizes, which corresponds to the exponential timer in our algorithm. Upon service completion, a job may leave the queue or re-enter it with some fixed probability which depends on its class. This is the queueing interpretation of the interruptions and resumptions of the algorithm.

The queue state is naturally described by the ordered sequence of job classes. We show that its stationary distribution does not depend on the detailed traffic characteristics beyond the traffic intensity of each class.

Network of processor-sharing queues. Looking at the detailed queue state is not relevant if we only want to observe the performance of the system in the long run. We consider instead an aggregate state, only retaining the number of jobs of each class in the queue. The stationary distribution of this state proves to be that of a network of processor-sharing queues (with one queue for each class in the cluster), as illustrated in Fig. 3 for the system depicted in Fig. 1. We also show that the average service rates in this aggregate state are allocated according to *balanced fairness* [3], a resource allocation which is known for its insensitivity property. These results are supported by simulations.

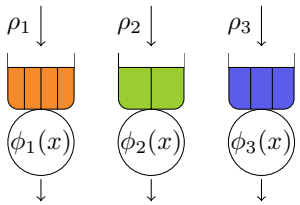


Fig. 3. Equivalent network of processor-sharing queues. Each queue corresponds to one class of jobs. The performance of this system only depends on the traffic intensity ρ_i of class i, for each $i = 1, 2, 3$.

IV. Conclusion

We have introduced a new algorithm, similar to round-robin scheduling, for computer clusters with compatibility constraints. We have assessed its performance with a new queueing model and showed that it achieves a balanced fair sharing of the computing resources. For the future works, we aim at gaining more insight on the impact of the mean number of interruptions per job on the sensitivity of the resource allocation.

References

[1] T. Bonald and C. Comte, "Balanced fair resource sharing in computer clusters," *CoRR*, vol. abs/1604.06763, 2017. [Online]. Available: https://arxiv.org/abs/1604.06763v2

[2] A. E. Krzesinski, "Order independent queues," in *Queueing Networks: A Fundamental Approach*, R. J. Boucherie and N. M. van Dijk, Eds. Boston, MA: Springer US, 2011, pp. 85–120.

[3] T. Bonald and A. Proutière, "Insensitive bandwidth sharing in data networks," *Queueing Syst.*, vol. 44, no. 1, pp. 69–100, 2003.

Elimination of network resilience impact in a video stream transport by implementing an SDN architecture constraint by existing network

Constant Colombo, Francis Lepage and Eric Gnaedinger
Université de Lorraine, France
CRAN, CNRS UMR 7039
Email: {constant.colombo,francis.lepage, eric.gnaedinger}@univ-lorraine.fr

René Kopp
TDF SAS, France
Network Architecture
Email: rene.kopp@tdf.fr

***Abstract*—In content delivery network, Quality of Experience provides major performance indicators such as availability and continuity of service. As a consequence, resilience has become a major concern for network operators. TDF operates a traditional transport network for video and audio transport through multicast. Any failure on the network causes a recovery time implying loss and an impact in the final content. This illustrate that service continuity is a direct consequence of network availability. This work aims to propose a Software Defined Networking (SDN) architecture in which the controller uses its knowledge of the performance and bandwidth allocation to compute multicast trees. Multicast trees are considered as Delay Constrained Minimal Steiner Tree (DCMST). In this paper, we proposed to deploy two link-independent trees carrying the same stream. This way, when a failure occurs, at least one of the trees is still active, eliminating any discontinuity on the final content.**

***Keywords*—Multicast; SDN; Resilience; Video stream; Audio stream; Delay Constrained; Steiner Tree.**

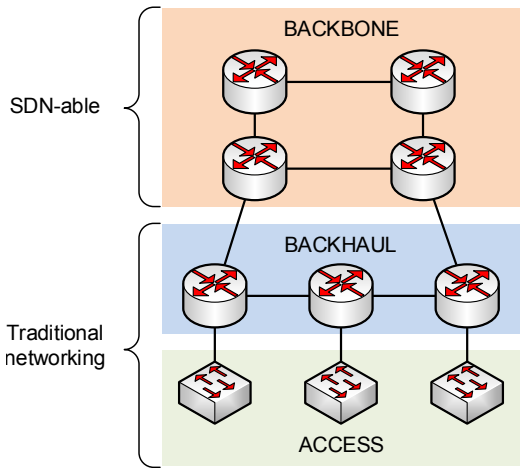


Fig. 1. TDF network hierarchy

I. Introduction

In content delivery network, resilience is one of the main problematic, as it is a directly responsible for network availability and thus service continuity.

TDF is a French network operator, mainly providing transport for audio and video streams nationwide. Today, its most important activity is multicast transport, streaming most of DTT ("TNT" in French) channels and national radios.

Given the carried traffic's nature, data cannot be retransmitted in case of loss and must be transported as fast and as steady as possible. Also, any loss in the network causes a multiplied impact in the final content, depending on the receiving decoder. As a consequence, the Quality of Experience (QoE) [1] is one of TDF's main concern. As service continuity is a consequence of network availability, Quality of Service (QoS) [2] is to be considered: commitments on availability, latency, jitter, and loss are specified for each customer in the Service Level Agreement (SLA).

Software Defined Networking (SDN) is a new paradigm in which the control plane is separated from the data plane in an entity called "controller". In other words, the controller handles decisions while network nodes only support forwarding, following the controller's instructions. This way, any functionality can now be implemented seamlessly through controller programming, and well known graph theory algorithms can be implemented in a centralised fashion.

This paper is organised as follow. In section II we describe the industrial context of this work. In section III we decompose the problematic in two child problems: how to compute an optimal tree and how to eliminate the impact of resiliency schemes. Section IV explains the chosen multicast model. Resiliency impact issue is addressed in Section V, describing the proposed scheme. Finally, section VI presents the next steps of this work.

II. Industrial Context

TDF's main network is a Hierarchical WAN called TMS (Transport Multi-Service), currently implementing MPLS over IP. It is composed of a layer 3 backbone, a layer 3 backhaul, and layer 2 access equipment, as illustrated by Fig. 1. We will see later that there are constraints regarding the SDN abilities that we wish to use in our approaches.

Today there are more than 800 nodes and 900 links in the network, mostly carrying multicast traffic for audio and video applications. The backbone and backhaul are mostly loops, so as to guaranty a possible path in case of failure, but the network cannot be considered fully-meshed.

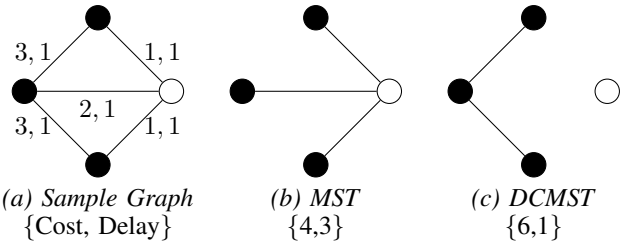


Fig. 2. Multicast Tree Solutions (Terminal nodes in black)

Given the variety of physical transmission mediums (Optical Fibre, Radio Frequency, Copper) and their different length, the network is heterogeneous in terms of latency, jitter and bandwidth capacity.

III. Problem Statement

When a failure happens in the network, detection mechanisms send a signal to the control plane, and routing protocols start to compute new routes to recover from it. Between the failure and the recovery, there are two periods -detection and recovery- during which traffic is interrupted.

Detection and recovery mechanisms in traditional networking have been largely studied and improved as shown in [3], [4], [5], [6], but both of them can never be reduced to zero. In other words, in case of rerouting, loss is unavoidable, and so is impact on the final content.

TDF current solution is Seamless Protection Switching [7]: link independent trees are deployed, carrying the same multicast stream through different paths. The destination equipment are able to recombine the two streams in one correct output, be it from direct video stream or IP. This way, in case of a link failure, one of the tree is still active, and the equipment's buffer can handle the switch between streams.

If this solution solves the problematic, there are still issues. In order for the trees to be link independent, they are designed and implemented by hand. It seems impossible to reach an optimal solution in terms of latency and bandwidth allocation this way. Plus, given the number of services carried by the network, only some of them benefit from this scheme.

The problematic can be divided into two child problems: first, how to compute an optimal multicast tree within our constraints, and then how to ensure the elimination of resiliency impact of this tree.

IV. Multicast Tree Model

The main multicast model in literature is the Minimal Steiner Tree (MST) solution. In a weighted graph, the Minimal Steiner Tree is the tree of minimal total weight connecting a set of nodes called terminal nodes, as shown in Fig. 2 (b). Terminal nodes are not necessarily leaves.

But as shown in [8] and illustrated in Fig. 2 (c), this might not be an optimal solution in our case: using delay as cost does give a minimal total cost, but the latency to each destination is longer.

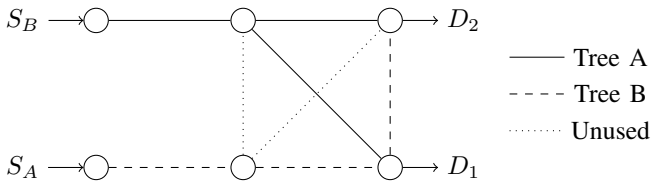


Fig. 3. Link Independent Trees

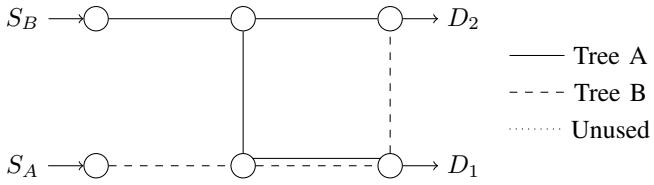


Fig. 4. Maximal Link Independent Trees

We chose to use a Delay Constrained Minimum Steiner Tree (DCMST), with bandwidth consumption as cost. In other words, instead of optimising both parameters, we optimise bandwidth consumption while staying under a latency limit. To solve the DCMST problem, many heuristics have been proposed such as in [9], and even an exact solution in [10].

V. Independent Trees

Our approach to eliminate the resiliency impact is to compute at least two link independent trees as illustrated in Fig. 3, to carry identical multicast streams. The two sources S_A and S_B send the same traffic to both of the destinations D_1 and D_2. In case of failure, either S_A or S_B is still able to carry the stream to destinations.

The point is to provide the destination a constant stream. In nominal situation, the destination equipment will receive two identical audio or video stream, and automatically discard any duplicates. In case of any failure, if the trees are independent enough, one of the tree should still be active. This way, the receiving equipment never loses any packet during the failure, and the final content is not impacted.

We aim to programmaticaly find two independent DCMST solutions, in the same fashion [11] describes path-rerouting. Also called "link-disjoint multipath routing" in [12], in which authors propose a scheme to compute independent Spanning Trees.

Considered networks might not be fully meshed, and as a consequence totally independent trees might not be feasible. We must be able to compute Maximal Independent Trees. The independence of the trees can be measured as the number of shared edges in the network. As illustrated in Fig. 4, the two trees have to share an edge in this topology. If this edge fails, both of the trees are disrupted. In this situation, one could object that a new tree can be computed. During this new tree establishment, there is a stream interruption, causing impact. Our main concern is not resiliency but its consequences. That's why we aim to reduce the common edges of the trees to a minimum.

VI. Future work

On the independent tree approach, our main future work is to find how to compute complementary DCMST solutions, in such a way that both of the trees present maximal independence.

We also have to further study the SDN constraints inherent to TDF's networks, according to available equipment and existing architecture, especially the hydrid SDN-traditional network problematic. Indeed, even if most of the core equipment support SDN features and implement OpenFlow [13], not all of TDF's network is SDN-able, as shown in Fig. 1.

Knowing these constraints, we will then propose an implementation on a controller and evaluate it. For all the advantages an automated network provides, we have to ensure that availability is improved and resilience impact is reduced. We plan on using Mininet [14] to generate a virtual topology of a random non-oriented 2-connected weighted delay constrained multigraph of various sizes.

Once our implementation is validated for a network the same scale as the industrial studied network, we will deploy it on real network.

References

[1] ITU-T,"G.1080: Quality of experience requirements for IPTV services", 2008.

[2] ITU-T, "E.800 Terms and definitions related to quality of service and network performance including dependability", 1994

[3] M. Pasin, S. Fontaine, S. Bouchenak, "Failure Detection in Large Scale Systems: a Survey", IEEE NOMS Workshops, 2008

[4] V. Chandola, A. Banerjee, V. Kumar, "Anomaly detection: A survey", ACM Computing Surveys, 2009.

[5] M. H. Bhuyan, D. K. Bhattacharyya, J. K. Kalita, "Network anomaly detection: Methods, systems and tools", IEEE Communications Surveys and Tutorials, 2014.

[6] M. Baingne, K. Palanivel, "Survey on Fast IP Network Recovery", IJCSIT, 2014.

[7] ST 2022-7:2013 - SMPTE Standard - Seamless Protection Switching of SMPTE ST 2022 IP Datagrams.

[8] J. Cho and J. Breen, "Is a Steiner Tree the optimal multicast model?", Monash University Clayton, vol. 3168, 1998.

[9] R. Forsati, M. Mahdavi, A. T. Haghighat, and A. Ghariniyat, "An efficient algorithm for bandwidth-delay constrained least cost multicast routing", Electrical and Computer Engineering, 2008.

[10] G. Feng, "Delay constrained multicast routing: What can we learn from an exact approach?", GLOBECOM, 2012.

[11] M. Médard, S. G. Finn, R. A. Barry, and R. G. Gallager, "Redundant Trees for Preplanned Recovery in Arbitrary Vertex-Redundant or Edge-Redundant Graphs", IEEE/ACM Transactions on Networking, 1999.

[12] A. Gopalan and S. Ramasubramanian, "IP Fast Rerouting and Disjoint Multipath Routing With Three Edge-Independent Spanning Trees", IEEE/ACM Transactions on Networking, 2016.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Palukar, L. Peterson, J. Redford, S. Shenker, J. Turner, *OpenFlow: enabling innovation in campusnetworks*, ACM SIGCOMM Computer Communication Review, 2008.

[14] B. Lantz, B. Heller and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks", ACM SIGCOMM Workshop on Hot Topics in Networks, 2010, http://mininet.org/

An object store for Fog infrastructures based on IPFS and a Scale-Out NAS

Bastien Confais
CNRS, LS2N, UMR 6004,
Polytech Nantes,
rue Christian Pauc, BP 50609,
44306 Nantes Cedex 3, France

Adrien Lebre
INRIA, LS2N, UMR 6004,
IMT-A,
4 Rue Alfred Kastler,
44300 Nantes, France

Benoît Parrein
Université de Nantes, LS2N, UMR 6004,
Polytech Nantes,
rue Christian Pauc, BP 50609,
44306 Nantes Cedex 3, France

***Abstract*—The Cloud Computing approach concentrates the computing power in few datacenters. The high latency to reach the platform makes this architecture not well suited for the Internet of Things. The Fog and Edge Computing propose to place servers near the users. In this context, we propose a first-class object store service for Fog/Edge facilities. Our proposal is built with Scale-out Network Attached Storage systems (NAS) and IPFS, a BitTorrent-based object store spread throughout the Fog/Edge infrastructure. Without impacting the IPFS advantages particularly in terms of data mobility, the use of a Scale-out NAS on each site reduces the inter-site exchanges that are costly but mandatory for the metadata management in the original IPFS implementation. Several experiments conducted on Grid'5000 testbed are analysed and confirmed, first, the benefit of using an object store service spread at the Edge and second, the importance of mitigating inter-site accesses. The paper concludes by giving a few directions to improve the performance and fault tolerance criteria of our Fog/Edge Object Store Service.**

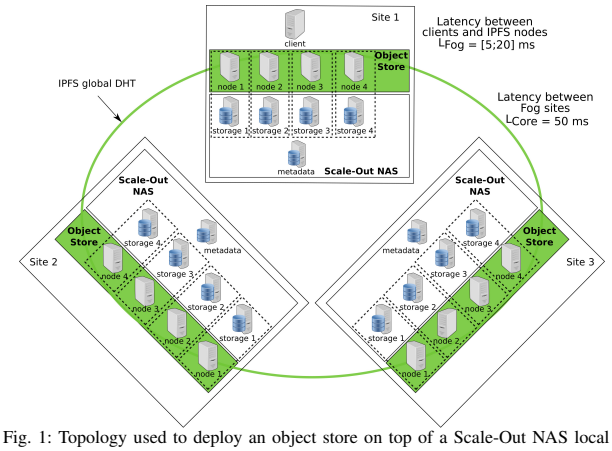


Fig. 1: Topology used to deploy an object store on top of a Scale-Out NAS local to each site.

I. Introduction

The Cloud Computing is not able to provide low latency computing that is necessary for the Internet of Things. The Fog Computing aims to deploy a lot of small datacenters geographically spread at the Edge of the network, close to the users to be reached with a quite low latency [1]. In this context, we are interested in the storage service that may be used in a such infrastructure. After having proposed a list of properties a storage system should have to be used in a Fog environment, we have shown that Interplanetary FileSystem (IPFS) was the best candidate to be used in a Fog environment [2]. Nevertheless, some improvements should be done such as reducing the amount of network traffic sent between the different sites of Fog.

In this work, we present our use of IPFS used on top of a Scale-Out NAS (Network Attached Storage) in Section II. Then, in Section III, we measure the performance of our approach on the Grid'5000 testbed not only by increasing the network latencies between the Fog sites but also by limiting the network throughput of the clients. Finally, Section IV concludes and gives some perspectives.

We limit our evaluation study to local access, when objects are stored on the closest site of the users.

II. Storage solutions

Interplaneraty FileSystem (IPFS) [3] is an object store based on a Kademlia distributed hash table (DHT) used to store the location of the objects and protocol similar to BitTorrent to transfer the data between the nodes.

The problem of IPFS when it is deployed in a Fog Computing environment is that the global DHT containing the location of every object does not provide any locality. When an object is requested from a node which does not store it, the node has to locate the object by sending a request outside the Fog site, even if the object is finally stored on another node located on the same site. The request sent outside the site is a major problem because it increases the latencies and avoids the users to access the objects stored on the site in case of network partitioning.

To solve this problem, we proposed an original idea consisting in deploying a distributed filesystem locally on each site. This filesystem is used by IPFS to store objects. The interest of this coupling is that objects stored in a site are available directly to all the IPFS nodes of the site. The DHT is only used to locate objects that are not stored on the local site.

Our solution avoids the interactions with the remote sites. A such deployment is illustrated in Figure 1, where all the clients are located on a same site. We show that the overhead related to the use of the distributed file system is quite low in writing. In reading, not to access the global DHT reduces significantly the access times.

		Mean **writing** time (seconds)			Mean **reading** time (seconds)		
	Workload \ Latency L_{Fog}	5ms	10ms	20ms	5ms	10ms	20ms
IPFS+ RozoFS	100×256KB	0.455	0.479	0.621	0.380	0.489	0.687
	100×1MB	1.541	1.585	1.782	1.401	1.493	1.894
	100×10MB	15.156	15.022	14.847	14.708	15.099	15.029
IPFS Seul	100×256KB	0.473	0.490	0.730	0.397	0.531	0.776
	100×1MB	1.647	1.644	1.855	1.403	1.515	1.934
	100×10MB	15.260	15.240	15.410	14.102	14.763	15.326

TABLE I: Mean time to read or write objects with a size of {256KB. 1MB and 10MB} in function of the L_{Fog} latency. Only the workloads using 100 objects are represented.

III. Experimentation

In this section, we compare the performance of IPFS used alone and IPFS coupled with the Scale-Out NAS RozoFS [4].

A. Material and Method

The evaluation of IPFS and IPFS coupled with RozoFS is performed on the Grid'5000 testbed. RozoFS is an open-source distributed filesystem providing good performance in both sequential and random access. The topology in Figure 1 is considered, composed of 3 sites. Each of them has 4 IPFS nodes which are also RozoFS nodes and a metadata server for RozoFS. All the clients (between 1 and 10) are connected to the same site of Fog and we focus only on the local access: objects are stored on the site the clients are connected to. The scenario consists for the clients to write all the objects on the site and to they read them. The IPFS node each request is sent to is selected randomly by the clients and all the requests are sent simultaneously. We measure the time to write and read the objects. We vary the number of clients and also the network latency between the clients and the IPFS nodes (L_{Fog}) between 5 and 20 ms [5], [6]. The network latency between the Fog sites is set to $L_{Core} = 50$ ms and the latency between the nodes of a same site is set to 0.5 ms. Latencies are emulated artificially thanks to the Linux Traffic Control Utility (`tc`). To get a more realistic scenario, we also limited the throughput of the clients to 512 Mbps that is close to the throughput of a LTE network [6]. Metadata replication in IPFS is disabled and data are stored in a `tmpfs` to remove the potential biases from the underlying storage unit.

B. Results

Table I shows the access time for an object in function of the L_{Fog} latency when only one client writes and reads 100 objects on its site. The main observation is that the latency has a bigger impact on small objects (256 KB) than big objects (10 MB). It takes 0.380 s to read an object of 256 KB when $L_{Fog} = 5$ ms and 0.687 s when $L_{Fog} = 20$ ms whereas it takes 14.708 s and 14.102 s with 10 MB objects. For the objects of 10 MB, the impact of L_{Fog} is small compared to the time to transfer the object. We also observe that the two solutions have similar writing times (1.782 s vs 1.855 s for objects of $1\,MB$ and $L_{Fog} = 20$ ms) Reading times are also similar except when small objects are used (256 KB). In this case, accessing the DHT becomes costly with a lot of objects

Figure 2 shows the access time of each object for a given client when a 10 clients read 100 objects of 256 KB. The L_{Core} latency is set to 200 ms (potential latency for a mobile Fog site) and L_{Fog} latency is set to 10 ms. We observe that

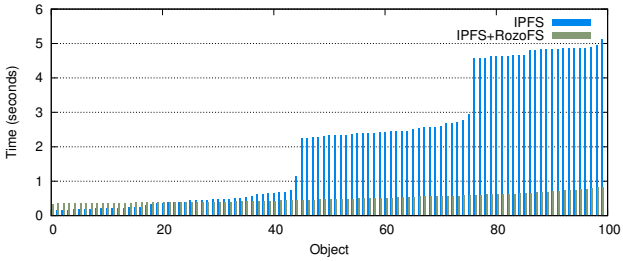


Fig. 2: Reading time of each object for a given client and a given iteration (100×256 KB).

access times are lower and more stable when IPFS is coupled with RozoFS (approximately 0.5 s). On the contrary when IPFS is used alone, access times vary a lot depending on the number of hops needed to be done to reach the node storing the location of the object in the DHT.

IV. Conclusion and Perspectives

We showed coupling IPFS to a Scale-Out NAS improves the local reading times without increase of the writing time. The coupling also limits the amount of network traffic exchanged between the sites and enables IPFS to work in case of network partitioning.

A future work may be to do the same test by considering access of objects stored on a remote site. We also consider replacing the global DHT used to locate the objects by the Locator/ID Seperation Protocol (LISP) and a Software Defined Networks (SDN) controller may be introduced to place each object on the most appropriated node.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12, 2012, pp. 13–16.

[2] B. Confais, A. Lebre, and B. Parrein, "Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures," in *IEEE CloudCom*, Luxembourg, Dec. 2016.

[3] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," Protocol Labs Inc., Tech. Rep., 2014.

[4] D. Pertin, S. David, P. Évenou, B. Parrein, and N. Normand, "Distributed File System based on Erasure Coding for I/O Intensive Applications," in *4th International Conference on Cloud Computing and Service Science*, Barcelone, Spain, Apr. 2014.

[5] K. Sui, M. Zhou, D. Liu, M. Ma, D. Pei, Y. Zhao, Z. Li, and T. Moscibroda, "Characterizing and Improving WiFi Latency in Large-Scale Operational Networks," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '16. New-York, USA: ACM, 2016, pp. 347–360.

[6] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A Close Examination of Performance and Power Characteristics of 4G LTE networks," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '12. New York, NY, USA: ACM, 2012, pp. 225–238.

Empowering Virtualized Networks with Measurement As a Service

Karyna Gogunska, Chadi Barakat
Université Côte d'Azur, Inria Sophia Antipolis, France

Guillaume Urvoy-Keller, Dino Lopez
Université Côte d'Azur, CNRS/I3S, France

***Abstract*—Our objective in this work is to devise a Measurement as a Service (MaaS) approach applicable in a data center (DC) with several tenants and one manager. Measurement might be done at the DC scale or be focused on a specific set of physical or virtual machines. As a first step towards this objective, we consider the case of a single physical server and the popular *sflow* measurement tool. We formulate the following problem: how can we adjust the sampling rate of *sflow* so as to obtain the maximum amount of information from the measurement setup without disrupting forwarding procedure of the operational traffic. We present our measurement set-up and our approach to address the problem.**

Keywords—sflow, network measurement, open vswitch, datacenter.

I. MOTIVATION

The virtualization of physical servers, disk resources and network functions constitutes a heavy trend in modern institutions, network providers and enterprises. Administrations consider transferring their internal data centers to a virtualized environment, where managing network infrastructure becomes more scalable, easier, cheaper and faster. In such an environment, networking hardware devices (switches, routers and middleboxes) are replaced by virtual machines embedding so-called network functions such as firewalls, load balancing and intrusion detection.

Debugging faults in this blend of physical and virtual network devices and servers requires a complex view on events and behavior inside these networks. This entails, as a first step, capturing and analyzing traffic without disrupting current switching of operational traffic of the tenants. This is the question that we tackle in this work at the scale of single physical server.

II. MEASUREMENT AS A SERVICE: THE SFLOW CASE.

In the present work, we consider the popular sflow tool, which enables to obtain flow level records with the additional ability to control the sampling rate of the packets out of which records are built [1]. We consider two cases. First, we want to obtain sflow measurements at the scale of a global physical server, i.e. for all the virtual machines of the server. Second, we want to measure a specific set of VMs inside a server.

Our goal is to determine how to adjust the sampling rate so as to maximize the amount of data collected while not compromising processing of operational traffic of tenants. Such a disruption is possible as with server virtualization, the network is now starting within the physical server with a virtual switch that interconnects the VMs together and to the physical network interface cards (NICs) of the server. Thus, VMs and the virtual switch share the hardware resources of the server.

We consider the case where measurements are done in the physical server and not on the physical switches that interconnect the physical servers as measuring in the physical network might prevent us from capturing inter VM traffic that occur within the server. The latter scenario is a classical one for multi-tier applications (e.g., a front-end Apache Web server and backend MySQL database server) or with data analytics tools such Hadoop or Storm where computation tasks are co-located on the same server.

As a first step to determine how to adjust the sampling rate, we aim at understanding the amount of resources, especially CPU, consumed by a virtual switch that is performing sflow collection at different sampling rates. Fig. 1 depicts the two scenarios we consider, with several virtual machines interconnected by an Open vSwitch (OvS) [2]. We chose OvS due to its popularity and the set of features that it offers, which includes sflow support. Traffic is generated between the VMs using flowgrind [3], as it enables to control the number of flows (along with other characteristics such as size) in a controllable manner, to gradually stress the sflow module of OvS. In Fig. 1 (a), we consider the case where we want to measure the traffic of all the VMs in the server and thus, we perform the measurement at the OvS deployed, e.g., by OpenStack, to interconnect the VMs and the physical NIC.

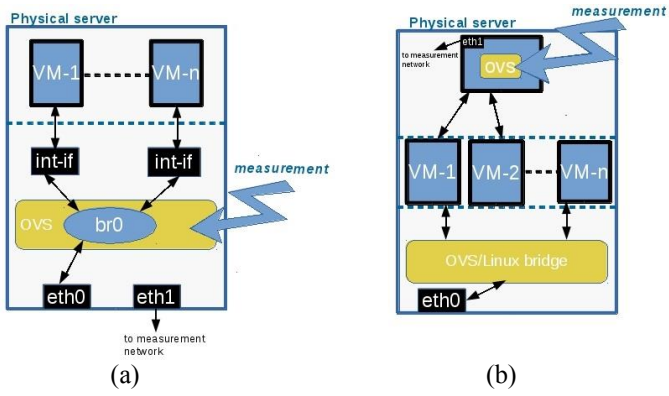


Fig. 1. (a) Global measurement. (b) Focused measurement

Fig. 1 (b) corresponds to the second scenario where measurement is targeted at a subset of the VMs. In this case, we advocate that using a dedicated virtual machine (a MaaS VNF) and redirect traffic to this VNF is a better option than measuring at the level of OvS on the physical server which operates the traffic of all the VMs in the network. This is because latter OvS does not allow to restrict sflow measurements to a specific set of ports.

We are in the process of collecting measurements at the moment and we hope to significantly progress towards devising rules to adjust the sampling by the time of the conference.

III. RELATED WORK

Several works have addressed the problem of monitoring and troubleshooting the performance of data centers. Hansel [4] and Gretel [5] are focusing on debugging faults in OpenStack [6]. To do so, they monitor the control plane of OpenStack, i.e., the control traffic between the different components (Nova, Neutron, etc.) to pinpoint the origin of problems.

As SDN is receiving a lot of attention and is a candidate to build next generation data center networks, a number of studies have focused on troubleshooting SDN networks. Monocle [7] and VeriDP [8] aim at checking the consistency between the SDN controllers and the rules actually implemented in the SDN switches, i.e. whether packet forwarding behaviors agree with the network configuration. With Dream [9], the authors propose to leverage SDN rule counters to implement measurement tasks through SDN rules. Watcher [10] provides a flexible and scalable resource optimization service for multi-tenant OpenStack-based clouds. UMON [11] is designed to provide flexible monitoring of user-defined traffic in SDN by decoupling monitoring from forwarding in OvS. As for our research, we though target flow level monitoring with sflow which is already implemented in OvS, by leveraging sampling feature to define manners (i) to prevent any overload of the virtual/physical equipment on which measurement is performed, and (ii) allocate resources so as to maximize the efficiency of a measurement task

ACKNOWLEDGMENT

This work was partly funded by the French Government (National Research Agency, ANR) through the “Investments for the Future” Program reference #ANR-11-LABX-0031-01.

REFERENCES

[1] sFlow. www.sflow.org

[2] Open vSwitch. www.openvswitch.org/

[3] Zimmermann A., Hannemann A., Kosse T. Flowgrind-a new performance measurement tool //Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE. – IEEE, 2010. – C. 1-6.

[4] D. Sharma et al. HANSEL: Diagnosing Faults in OpenStack. In CoNEXT’15.

[5] Goel A., Kalra S., Dhawan M. GRETEL: Lightweight Fault Localization for OpenStack //Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies. – ACM, 2016. – C. 413-426.

[6] OpenStack. https://www.openstack.org/

[7] M. Kuzniar, P. Peresini, and D. Kostic. Monocle: Dynamic, fine-grained data plane monitoring. In ACM CoNEXT, 2015.

[8] Zhang P. et al. Mind the Gap: Monitoring the Control-Data Plane Consistency in Software Defined Networks //Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies. – ACM, 2016. – C. 19-33.

[9] Moshref M. et al. DREAM: dynamic resource allocation for software-defined measurement //ACM SIGCOMM Computer Communication Review. – ACM, 2014. – T. 44. – №. 4. – C. 419-430.

[10] Watcher. https://wiki.openstack.org/wiki/Watcher

[11] Wang A. et al. Umon: Flexible and fine grained traffic monitoring in open vswitch //Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies. – ACM, 2015. – C. 15.

Software-Defined Real-Time Mesh Networking: Protocol and Experimentation Method

Florian Greff*†, Ye-Qiong Song†, Laurent Ciarletta† and Arnaud Samama*
*Thales Research & Technology - Palaiseau, France
{florian.greff — arnaud.samama}@thalesgroup.com
†LORIA Research Lab - University of Lorraine - Nancy, France
{florian.greff — ye-qiong.song — laurent.ciarletta}@loria.fr

***Abstract*—Our research address the problem of network resource allocation for real-time communications in mesh networks. We are studying the combination of online flow admission control and pathfinding algorithms in an SDN-like controller. We propose a Software-Defined Real-time Networking protocol that allows a dynamic and incremental allocation of network resource in a mixed-critical real-time mesh network context. We also propose a method to ensure the dependability of the network, considering that transmission errors and node failures may happen. Finally, our research includes the study of a new way of experimenting on embedded networks, by making use of both an in-silicon platform and a simulator. We designed an original framework to ease the driving of such experiments.**

I. SDRN Protocol

Nowadays, in embedded real-time systems, there is an evolution of needs in terms of computing power and communication between applications. This is due to the changing qualities of sensors, whose data processing needs are increasing (for example in radar applications), and the emergence of new application categories such as multi-sensors. Interaction between applications and sensors is increasing, while features tend to be spread over several computing units. Communication architectures are becoming more complex.

Mesh networking of the components of such systems would reconcile their real-time constraints with the new application needs. The plurality of communication paths should result in increased flexibility, resilience, scalability and load balancing characteristics. To improve overall system resilience, it must be possible to redeploy a processing function running on a faulty computing unit to a non-faulty one, or operate a graceful Quality of Service (QoS) degradation [1].

From the networking point of view, this implies the ability to dynamically allocate network resources with respect to the needs of running applications. We define this kind of platform as a Software-Defined Real-time Network (SDRN). Applications may express, at runtime, their needs in terms of real-time and bandwidth constraints. The SDRN controller performs an admission control for each declared communication and makes sure that all the accepted flows are meeting their constraints, through a proper network configuration. To achieve this, we have to answer three questions:

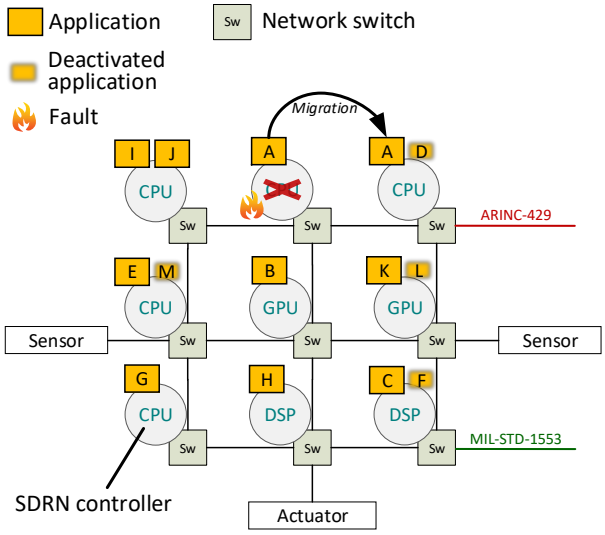


Fig. 1. Dynamic mesh networking of components and applications

1) Which network mechanisms do we need in order to guarantee that real-time constraints are respected, while ensuring that the set of allocated flows can evolve at runtime?
2) How do we perform admission control?
3) How do we handle a link or node failure?

Our work focuses on these three questions, to which we add further requirements:

- No need for clock synchronization, for scalability purposes.
- Realistic admission computation time to make online reconfiguration possible.
- A protocol with the best possible overall bandwidth utilization, i.e. the least possible waste of bandwidth due to non-utilization of resource and due to overheads.

Once the required network mechanisms have been defined, the low-level network layer can be designed using technologies

such as Ethernet AVB or Time-Sensitive Networking [2], RapidIO [3], PCIe [4], etc. For our study, we have chosen the store-and-forward RapidIO technology to build our experimentation platform, described in [5]. However, the SDRN solution is designed to be usable on top of COTS (Component Off The Shelf) protocols.

Our contributions are:

- A network resource dynamic allocation protocol, SDRN [6]. It ensures the respect of all real-time constraints once the nodes have been configured by a SDN-like controller. It also takes packet fragmentation – by the underlying protocol – into account, which enables to use it on top of any kind of network, whatever the format of the low-level frame. The protocol is based on link level virtualization for flow isolation.

- A combined admission control and pathfinding algorithm for new resource requests. The SDRN controller is responsible for analyzing, at runtime, the resource requests from applications and find (if they exist) optimal paths relative to these requests. It also configures the network components with respect to its decisions.

- A method to handle fault tolerance, in order to ensure a Mean Time Between Failure for the whole platform. We use a probabilistic approach to take increased delays due to retransmission of packets into account, and a hybrid routing to handle transient periods after a link or node failure.

The novelty of the SDRN protocol does not lie in pure timing analysis or optimal pathfinding, which are separate well-researched topics, but in adressing their combination to achieve a dynamic allocation of heterogeneous real-time communication flows on a mesh network.

II. ERICA Experimentation Framework

Experimentation is an important part of the design process of new network mechanisms. It can be performed via the use of simulation, emulation or physical testbeds. Each approach brings advantages and disavantadges in terms of accuracy, flexibility, scalability, repeatability and instrumentation. In many cases, the quality of evaluation is linked to the ability to provide realistic application behaviors. When experimenting on a physical testbed, this realistic behavior can basically be achieved by running real applications on top of the physical nodes. When experimenting on a simulator, there are typically two ways of doing it. The first one is simulating the applications, which is either at the cost of a loss of accuracy, or time and complexity if the whole high layer (including the CPU, OS, etc.) is to be simulated. Moreover,

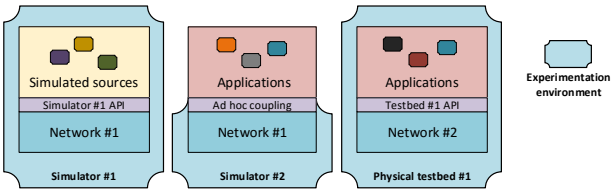


Fig. 2. Environment-driven standard approach

it may be hard to distinguish which differences between the observed and expected results come from the network layer and which come from the application one. Another solution is to run real applications over the simulated network. This is close to the Hardware-in-the-Loop approach, but at the application level. This implies using or developing – often *ad hoc* – interfaces in order to couple the applications with the simulator. However, each interface is related to a given environment (a given simulator for example). This is why we consider current approaches as environment-driven (Figure 2), which means that it is often impossible to reuse or transfer the part of an experimentation stack from an experimentation environment to another one.

These considerations introduce the need for a framework to provide adaptation mechanisms in order to run an application layer on top of different kind of experimentation networks (either real or simulated), thus maximizing relevance, portability and reusability.

ERICA [5] which stands for ExpeRImenting Cross-layer real-time QoS for Autonomic systems, was initially designed to enable the use of real benchmarking applications over both a physical platform and a network simulator. This is what we call a hybrid experimentation stack. As detailed in [5], this architecture allows us to design a new kind of network through a symbiotic approach. This approach consists in alternatively making use of the physical platform and the simulator, in a iterative way, to ensure the relevance of the simulation from which we design new network mechanisms. In this work, we felt the need for a higher-level interface to enable an agile switching between the platform and the simulator, while keeping the same real application layer.

From this background, we extended the initial ERICA framework into a more global one, designed to ease conducting hybrid experiments, by enabling the use of real applications whichever the underlying network (Figure 3). A high-level interface allows the user to draw the desired experimentation network topology, place applications on components, then automatically generate all what is needed to conduct the experiment: *ad hoc* simulation actors, configuration files, a window to launch applications, etc. ERICA experimentations can thus be seen as Software-Defined Experimentations. Furthermore, the framework can be extended with the support of

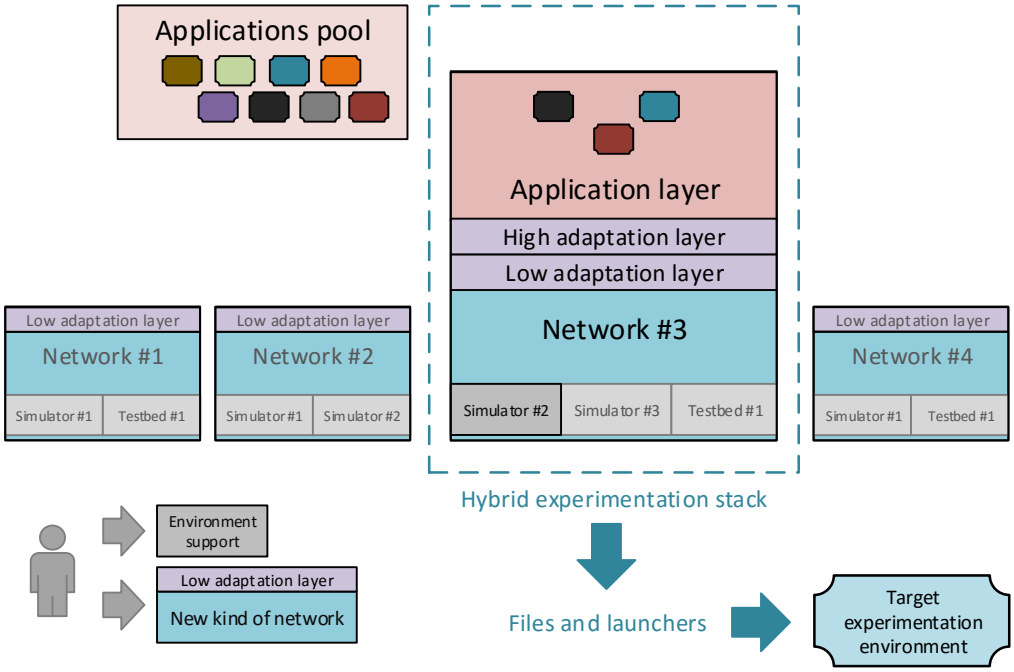


Fig. 3. ERICA experimentation architecture

a new kind of network or new experimentation environment (software extensions). Once a plugin has been integrated, all the complexity of the related experimentation layer stays transparent from the user point of view. This enables the reusability and portability of hybrid experimentation methods.

REFERENCES

[1] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, 1999.

[2] M. D. J. Teener *et al.*, "Heterogeneous networks for audio and video: Using IEEE 802.1 audio video bridging," *Proceedings of the IEEE*, vol. 101, no. 11, 2013.

[3] RapidIO Trade Association, "RIO 3.1 Specification," 2014.

[4] PCI Special Interest Group, "PCIe 3.1 Specification," 2014.

[5] F. Greff *et al.*, "A Symbiotic Approach to Designing Cross-Layer QoS in Embedded Real-Time Systems," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[6] F. Greff *et al.*, "A Dynamic Flow Allocation Method for the Design of a Software-Defined Real-Time Mesh Network," in *13th IEEE International Workshop on Factory Communication Systems (WFCS 2017)*, 2017.

A-TLSA: An Adaptation of the Two-Level Scheduling Algorithm from WiMAX to LTE Networks

Florian Leman
LS2N
CNRS UMR 6004,
Facultés des Sciences et des Techniques,
2 rue de la Houssinière
44300 Nantes FRANCE
Email: florian.leman@univ-nantes.fr

Salima Hamma
LS2N
CNRS UMR 6004,
Facultés des Sciences et des Techniques,
2 rue de la Houssinière
44300 Nantes FRANCE
Email: salima.hamma@univ-nantes.fr

Benoît Parrein
LS2N
CNRS UMR 6004,
Polytech'Nantes,
rue Christian Pauc - BP 50609 -
44306 Nantes Cedex 3 FRANCE
Email: benoit.parrein@univ-nantes.fr

***Abstract*—Scheduling in LTE (Long-Term Evolution) is a challenging functionality to design, especially in the uplink. We propose in this paper an adaptation of TLSA (Two-Level Scheduling Algorithm) which have been proposed in Worldwide Interoperability for Microwave Access (WIMAX) in LTE context. We first propose a mapping service classes between WiMax and LTE and then propose the bandwidth allocation strategy inter-classes and intra-classes. It's insure first QoS for all service classes avoiding starvation of lower priority classes (i.e. Best Effort). In the second time, it ensures fair bandwidth allocation in each class.**

I. Introduction

In recent years, LTE Long Term Evolution, also known as 3G has became the main technology to allow mobile internet access on smartphones. The 3GPP (3rd Generation Partnership Project) has introduced the LTE (Long-Term Evolution) specifications as the next step for the 4G/5G cellular networks. The main objectives of LTE is to support several hundreds of active users per cell, to reduce user plane latency, to support connection with several antennas simultaneously and to be able to adjust the width of the spectrum used. Thus, scheduling LTE's uplink requires particular attention. It's plays a crucial role as it manages the limited radio resources at LTE's access level.
Scheduling in LTE is managed on the level of the base station called eNB (evolved NodeB) and operates within LTE's MAC layer. It's responsible for allocating shared radio resources among mobile User Equipments (UEs). The intelligence of the eNB is associated with awareness of network conditions such as wireless channel quality and the QoS experienced by the diverse Internet services running over the LTE interface.

II. TLSA

TLSA Two-Level Scheduling Algorithm is an uplink scheduling algorithm originally designed for WiMAX [1]. In this work, we have attempt to adapt it to LTE. The algorithm works as follow : at the first level, the bandwidth is distributed among the classes of traffic. It guarantees QoS requirements, and prevents starvation from lower priority classes. At the second level, class-specific algorithms are used to distribute the class allocated bandwidth among the active connections of the same, with the aim to ensure fair resources allocations among flows of the same class.
The First Level Scheduling (FLS) must ensure the following conditions : (i) QoS is ensured for all classes of traffic ; (ii) lower priority flows could not affect higher priority flows ; (iii) lower priority traffic is not starved ; (iv) high bandwidth utilisation.
The Second Level Scheduling varies according to the class as we will see later.

III. Adaptation

Adapting this algorithm from WiMAX to LTE is an harsh task, as there are huge differences between these two technologies:

- Classes of traffic are not the same.
- The buffer status reports work in a different way : for WiMAX, each flow reports its own buffer status, whereas in LTE, flows are packed into a group, and only the buffer status of the groups are reported. Moreover, in WiMAX the value reported is a new bandwidth request, whereas in LTE it's a raw value of the buffer (ie. a previous request can be overlaped in a new buffer status value). LTE defines only 4 groups of flows, numbered from 0 to 3 (the lower is the number, the higher is the priority)'
- LTE's transmission time interval (TTI) lasts 1 millisecond, whereas WiMAX's one is 20 milliseconds.

The first point seen above is probably the more important. As the classes of traffic are slightly different, we have to find an equivalence between WiMAX's classes and LTE's ones. WiMAX uses 5 different classes :

- UGS Unsollicited Grant Services which is a constant bit-rate and delay constraint class.

- eRTPS Extented Real Time Polling Service which have a definition really close to UGS.
- RTPS Real Time Polling Service which is a delay constraint class.
- NRTPS Non-Real Time Polling Service which has a minimum througput guaranteed.
- BE Best Effort a class without any specific requirement.

LTE uses 9 different classes [2] (further definition defines 4 additional classes) :

- Classes 1 and 2 are for conversational voice and video (delay constraint, guaranteed bitrate).
- Class 3 is for real time gaming (high delay constraint).
- Class 4 is for non-conversational video (buffered streaming) (guaranteed bitrate).
- Class 5 is IMS signalling (delay constraint, non-guaranteed bitrate)
- Class 7 is for voice, video, live-streaming, interactive gaming (but non-guaranteed bitrate)
- Classes 6, 8 and 9 are for all other usages, TCP-based connections (no delay constraint, non-guaranteed bitrate). The only difference between these classes is that users of class 6 have a higher priority on users of class 8, whose have a higher priority on users of class 9.

For A-TLSA, we propose the following matching:

- UGS and eRTPS are mapped to classes 1, 2, and 5 in group 0. We have added the class 5 in this group, despite it's not a GBR Guaranteed BitRate because the delay constraint on class 5 is high and it's a low througput class.
- RTPS is mapped to class 3 in group 1, as there are delay constraint classes.
- NRTPS is mapped to class 4 in group 2, because only GBR classes in LTE report their bitrate at the connection setup event, and we need to know that value to ensure a minimum guaranteed.
- BE is mapped to classes 6, 7, 8, 9 in group 3, as they are all the remaining classes.

We have also changed the way RTPS (group 1 for LTE) does its intra-class distribution. In WiMAX, the TLSA algorithm uses a service ratio for each flow, and allocates a flow only if this service ratio is below the global service ratio. This ensures the fairness inside the rtPS class. The algorithm also keeps track of packet deadlines and makes sure they does not miss it. The service ratio can be roughly defined as the bytes requested over the bytes allocated. First, because buffer status reports raw value in LTE, we have redefined it to bytes requested over bytes requested plus bytes allocated. With this new equation, we have the equivalence for the intra-class distribution. But moreover, we have redefined it in terms of PRB Physical Resource Blocs , because the algorithm finally allocates PRB and not raw bytes. This give us a more precise allocation scheme, bringing an higher global throughput.
Other intra-classes distribution have not changed. Group 0 is allocated according to the requested bitrate ; Group 2 is allocated according to a proportion of the total amount of bytes requested by all nRTPS flows ; Group 3 is allocated with a round-robin algorithm (this will be probably changed for a fairlier one in the future).

IV. Simulation

The algorithm has been implemented under NS-3 [3], an open source discrete event network simulator written in C++. Simulations show that the awaited mecanism works correctly. However, no comparison with other algorithms have been done for the moment, this will be a future work.

V. Conclusion

Adapting the TLSA algorithm to the LTE is not an easy task, as the differences between WiMax and LTE are really important, especially the mecanism of buffer reporting. The solution proposed here keeps the internal logic of the algorithm, while innovating on the allocation using directly PRB units. The resulting algorithm is working, there are some features we will develop in a near future :

- WiMAX algorithm originally does not take into account link quality status. Moderm schedulers have absolutely to take it into account, as it's en essential feature reflecting the real conditions.
- WiMAX has a frame duration of 20 milliseconds, but LTE's one is really shorter (1ms for each subframe), so we will review the minimum guaranteed for group 3 to allocate it over several subframes, saving a lot of headers and increasing the global througput (when you allocate consecutive blocs of resources the average byte count by bloc increases).
- Compare our algorithm with other ones.

The rapid development of Internet of Things - IoT - brings also new problematics. NB-IoT - NarrowBand - and LTE-M - Mobile - are two emerging technologies designed to include IoT communications over the 3G/4G technologies. IoT has many types of connections : the delay contraint, the throughput vary in accordance with the kind of objects. By example, for small objects, communications have an ultra-low intermitent throughput, but aspects such as power management become predominant. Thus, integrating this kind of communications to our scheduling algorithm is a new challenge to be achieved in the future...

References

[1] Z. Ahmed and S. Hamma, *Two-Level Scheduling Algorithm for Different Classes of trafic in WiMAX Networks*

[2] 3GPP, *LTE release 10*. September 2011. http://www.3gpp.org/specifications/releases/70-release-10.

[3] *NS-3 Website*. October 2016. https://www.nsnam.org/.

A First Step Towards Security Extension for NFV Orchestrator

Montida Pattaranantakul[1,2], Yuchia Tseng[3], Ruan He[4], Zonghua Zhang[1], and Ahmed Meddahi[1]
[1]IMT Lille Douai, Institut Mines-Télécom, Lille, France
[2]Institut Mines-Télécom/TELECOM SudParis, Évry, France
[3]Paris Descartes University, Paris, France
[4]Orange Labs, Châtillon, France

***Abstract*—Network Functions Virtualization (NFV) has recently emerged as one of the new networking paradigms to significantly change the way that the networks and services are deployed, managed, and operated. One of the major advantages of NFV is to reduce hardware cost, meanwhile increasing service agility and scalability. Recently, there are many platforms for NFV management and orchestration (MANO) are available, however few of them contains dedicated modules or components for security management. This paper is intended to study the feasibility of extending the current NFV orchestrator to have the capability of managing security mechanisms. To do that, we propose a security extension module based on TOSCA data model which is commonly used by NFV MANO architecture. We then develop an access control use case to illustrate the usage of our proposed security extension. Specifically, we integrate the security extension into the Moon framework, which can automatically verify security attributes, generate access control policies, and further enforce the policies through the underlying infrastructure according to the high-level security policies. The preliminary results show that our security extension can work together with the NFV orchestrator to enable fine-grained access control to protect resources and services.**

I. Introduction

The nature of NFV allows to create and deploy new services more quickly, by decoupling network functions from proprietary hardware appliances, and implement them using software based approach. To do that, orchestration module is required to dynamically coordinate the underlying infrastructure resources. Therefore, NFV orchestration plays an important role in maintaining full lifecylce management of Virtual Network Functions (VNFs) and end-to-end network services, such as instantiation, configuration, termination, scale-in/out, resource and policy management, performance measurement, event correlation, validation and authorization of resource requests.

Although NFV has generated significant interests in the market place and academia, security concerns remain to be significant barrier to the wide adoption of NFV [1], [2]. In addition to the fact that the operators have to deal with non-trivial service complexities, spanning from configuration and maintenance to management and orchestration, in order to achieve NFV goals [3], [4], [5]. While the ETSI has published a document about NFV Management and Orchestration (NFV MANO) [6], a detailed description about NFV specification, service deployment, and security management is still missing. In particular, two critical issues are highlighted.

First, as a matter of fact, the current research activities are mainly focused on how to migrate network functions from dedicated hardware to virtualization environment, and how to manage and orchestrate the virtualized network functions on demand. The core concept of deploying and operating network services in NFV is based on service templates. These templates define the attributes and requirements to allocate resources and initiate network services, so that NFV orchestrator can use these templates as reference for further deployment. Therefore, the service templates must be modeled clearly and should be defined based on model-driven approach, which allow operators to obtain a clear view about the structure of network services, nodes, and links to be deployed. An appropriate data model can also facilitate the operators to gain visibility and controllability over the workloads, the deployed VNFs, and the underlying infrastructure resources. It is worth noting, however, many existing NFV frameworks from industrial sectors like [7], [8] do not support model-driven NFV orchestration, neither TOSCA data model standard [9].

Second, along with the orchestration of network functions and services, appropriate security mechanisms are expected to be put in place, which however is believed to be a major challenge considering the complexity of the processes of NFV management and orchestration. Moreover, the current NFV orchestrator is mainly based on ETSI NFV reference architecture, which aims at managing VNF lifecycle and orchestrate infrastructure resources for supporting end-to-end network services. That says, a typical NFV orchestrator does not necessarily contain the capability of managing security mechanisms, even though some of them reserve specific fields for specifying security attributes. To date, a best practice recommendation for implementing NFV based security management and orchestration has not yet appeared. Nevertheless, many existing NFV orchestration platforms especially the open source ones are based on TOSCA data model, none of them has been defined from security perspective.

To address the aforementioned issues, our contribution in this paper is three-fold.

- First, we conduct a comprehensive study on the existing NFV orchestration platforms, especially the open source ones, with a purpose to abstract their data models, analyze their operations and workflows, and eventually identify the capability of managing security mechanisms.
- Second, based on the understanding of NFV orchestration data models, we extend the ones using standard TOSCA data model with security aspect, by incorporating security attributes, enabling administrators to define security policy and allowing users to specify security attributes for each VM/VNF.

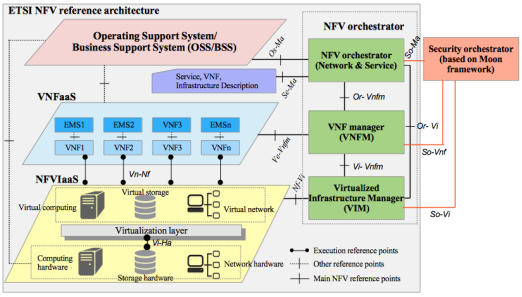


Fig. 1. High level architecture of security extension for NFV orchestrator aligned with ETSI NFV MANO specification

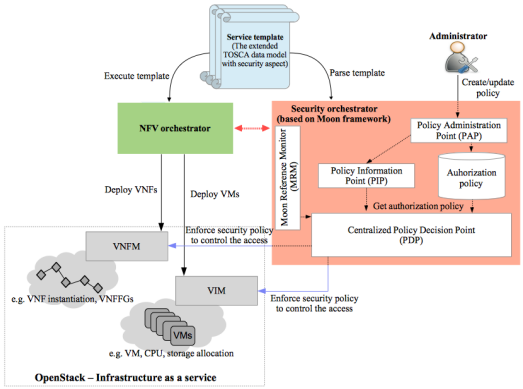


Fig. 2. The detailed model of security extension for NFV orchestrator

- Third, we develop a TOSCA based security extension by employing a well-developed security policy engine (Moon framework that is developed for access control) to achieve the capability of managing security mechanisms. Our extension can automatically verify security attributes specified in the extended TOSCA data model, generate corresponding access control policies, and enforce them to protect the NFV infrastructure.

II. Extending NFV Orchestrator for Security Management

A. Design Architecture

Fig. 1 presents a high level view of our security extension, which complies with ETSI NFV MANO architecture and works along with NFV orchestrator to provide automatic security control, verify security attributes, and enforce security policies.

B. Design Principle

The development of security extension contains two major steps. First, we extend TOSCA data model (simply known as service template) to use it for specifying high-level security policy and achieving fine-grained access control. For example, network nodes can be specified with certain security attributes and a group of security policies, so that only authorized requests that belong to the same group of security policy can gain access. Second, security functions such as access control are provided on demand, ensuring that the deployed resources/services are protected from unauthorized access based on the specified security policy. For better illustration, Fig. 2 shows the detailed model of security extension, which contains three main components,

- *TOSCA data model (or service template):* which defines data structures for NFV orchestration by using YAML [11]. It describes the deployment, operational behavior requirement, and link connection for each network service in NFV. Its structure contains nodes (e.g, VNFs) and relationships (e.g, Virtual Links (VLs)) defined together to deliver end-to-end lifecycle operations of network services. This template is treated as a YAML file, and stored in a catalog during the service onboarding process for service instantiation. An example TOSCA data model is shown in Fig 4: the left side of the figure represents a general structure, and the right side is about deploying VNFs. Our contribution here is to extend the typical TOSCA data model with security policy specification, in which all nodes presented in the template are defined with their security attributes or a group of security policies.
- *NFV orchestrator:* which maintains the lifecycle management of infrastructure resources and network services. NFV orchestrator uses the extended TOSCA data model (service template together with security policy and attribute specification) to allocate infrastructure resources (e.g, VMs, VLs), and instantiate virtual components (e.g, VNFs) at runtime based on the predefined relationship between these components. As indicated in Fig. 2, the NFV orchestrator interacts with Virtualized Infrastructure Manager (VIM) to allocate VMs, and interacts with VNF Manager (VNFM) to instantiate the VNFs.
- *Security extension:* which works together with NFV orchestrator to provide monitoring, control resource access, and enforce security policy to the deployed VMs/VNFs. Specifically, it uses the extended TOSCA data model to obtain security attributes of each VM/VNF, creates security policy, and enforces access control. To enforce the security policies, security orchestrator works with Moon framework - a security policy engine with attribute-based policy specification. If an authorized entity (*subject*) meets with the policy rules defined in a security policy, it is then allowed to access (*action*) the requested entity (*object*). Otherwise, the request is rejected. For example, if the deployed VNF is specified with a group of security policy, then only legitimate requests in the same security group are allowed to access to it. Other requests that are not from the same security group will be rejected.

C. Moon framework

Our security orchestrator is based on Moon framework [10], which is designed to monitor, control, and manage VMs/VNFs over the OpenStack infrastructure. The core part of current version Moon framework relies on an access control model, which specifies high-level security policies to determine authorization requests.

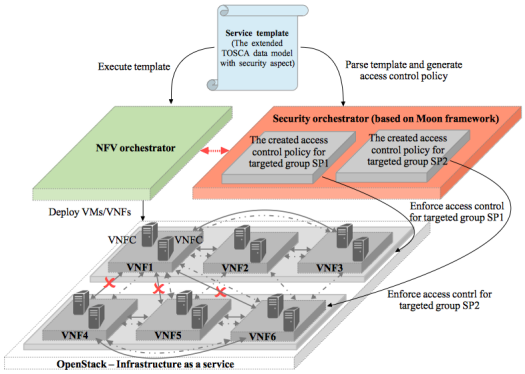


Fig. 3. Data model of security extension for NFV orchestrator, with "security policy specification" and with "no security policy specification" scenarios, respectively represented as dotted and solid lines

III. Use Case: Access Control

A. Scenario 1: Data model without security policy specification

In this scenario, we define TOSCA data model without any security policy specification. Thus, NFV orchestrator simply reads the descriptions of VNFs and deploys them accordingly. That says, security extension will not perform any access control, hence the predefined TOSCA data model does not specify any security policy for each VM/VNF. As a result, any access requests will be granted to access the infrastructure resources. An example scenario is given in Fig. 3 with dotted line. Clearly, this scenario does not ensure a trustworthy environment, and the fine-grained access control to the deployed resources/services is not available.

B. Scenario 2: Data model with security policy specification

In this scenario, TOSCA data model is extended to include security policy specification. Like the first scenario, NFV orchestrator deploys VM/VNF according to the template description of the extended TOSCA data model. Moreover, the security extension invokes the API of Moon to generate an access control policy based on security attributes extracted from the extended TOSCA data model. The predefined security attributes of each VM/VNF are assigned to the *subject* and *object* according to the specification of attribute-based access control policy. When a VM/VNF instance is executed, access request will be redirected to the security extension, which then enforce the corresponding access control policies.

This scenario is illustrated in Fig. 4, in which the data model defines six different types of services, respectively running on VNF1 – VNF6. Thus, two groups of security policy (highlighted in grey) are specified as follows, (1) SP1 provides VNF1, VNF2, and VNF3, and, (2) SP2 provides VNF4, VNF5, and VNF6. With such security policy enforced, VNF1 can gain access to VNF2 or VNF3, and vice versa, while none of the VNFs provided by SP1 can access to VNF4/VNF5/VNF6. Similarly, VNF4 can access the VNF5 or VNF6 and vice versa, while it can not access to VNF1/VNF2/VNF3.

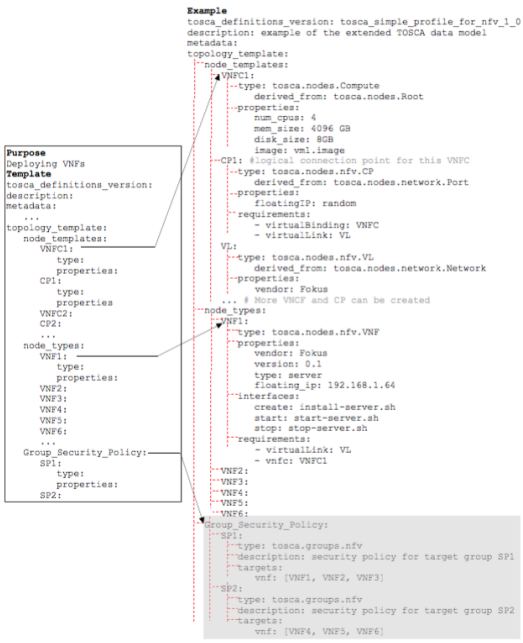


Fig. 4. The extended TOSCA data model with security policy specification (highlighted in grey)

When the extended TOSCA data model is passed to the security extension, a verification process is initiated to check whether the deployed VNFs have been specified any security attributes. Then the security extension invokes the API of Moon framework to generate appropriate access control policy by assigning security attributes of each VNF with *subject*, *object*, and *action*. The results are shown in Fig. 3 with solid line, which indicate that the deployed VNFs are permitted to access to each other if they have the same group of security policy, while the cross group requests are denied. The example clearly demonstrates that thanks to the security extension, the operators are able to enforce fine-grained access control to their assets by simply defining security attributes in the VNFs.

References

[1] Montida Pattaranantakul, Ruan He, Ahmed Meddahi, and Zonghua Zhang, "SecMANO: Towards Network Functions Virtualization (NFV) based Security MANagement and Orchestration", *IEEE TrustCom' 16*, Tianjin, China, Aug 2016, pp. 598-605.
[2] Ashutosh Dutta, "Security Challenges and Opportunities in SDN/NFV Networks", Nov 2016.
[3] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee, "Network Functions Virtualization: Challenges and Opportunities for Innovations", *IEEE Communications Magazine*, Feb 2015, Vol. 53, No. 2, pp. 90-97.
[4] Jorge Carapinha and others, "Network Virtualization - Opportunities and Challenges for Operator", *FIS' 10*, 2016, pp. 138-147.
[5] Rashid Mijumbi, Joan Serrat, Juan Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba, "Network Function Virtualizations: State of the Art and Research Challenges", *IEEE Communications Surveys & Tutorials*, Vol. 18, No. 1, pp. 236-262, 2016.
[6] ETSI GS NFV-MAN 001, "Network Functions Virtualization (NFV); Management and Orchestration", Dec 2014.
[7] CloudNFV, Jan 2014. [Online]. Available: http://www.cloudnfv.com/
[8] Alcatel-Lucent, "CloudBand", 2014.
[9] TOSCA, "TOSCA Simple Profile for Network Functions Virtualization (NFV version 1.0)", Mar 2016.
[10] OPNFV, "Moon - A Security Management System", Apr 2016. [Online]. Available: https://wiki.opnfv.org/display/moon/Moon
[11] YAML, "Ain't Markup Language", May 2001.

The AIS (Automatic Identification System) messages as big data and its applications in maritime field

Aboozar Rajabi, Claude Duvallet
Normandie Univ, France
ULH, LITIS, F-76600 Le Havre, France
{aboozar.rajabi, claude.duvallet}@univ-lehavre.fr

***Abstract*—In maritime-related fields, AIS (Automatic Identification System) messages are used to send data information on ships. The numerous amount of messages sent by the ships all over the world make us to consider them as a kind of big data. It has many advantages such as managing maritime traffic and decreasing environmental impacts. CIRMAR (CIRculation MARitime) is a research project aims at providing a proper platform to store and analyze these messages combining with some other related information. In this brief paper, this project, its different phases and applications and also future works are described in summary.**

Keywords—Big data; Automatic Identification System; maritime traffic; research platform;

I. INTRODUCTION

Nowadays, the amount of data in our world is exploding and analyzing large data collections – so called big data – is unavoidable and has tremendous benefits. In maritime-related fields, the AIS (Automatic Identification System) messages could be considered as big data due to the huge amount of data transmitted every day. Therefore, a proper platform is needed to store and analyze AIS messages.

CIRMAR (CIRculation MARitime) is such a research platform using AIS data based on affiliation to a collaborative network. It provides required processing and analyzing tools to overcome various methodological and technical issues of using AIS messages. For example, the duration of observing AIS messages must be long enough to make seasonal trends reliable or its geographic scope must be comprehensive to explore the economic gap among different regions of the world.

The remainder of this abstract paper is organized as follows: Section II presents an overview of the AIS. The CIRMAR project with its different phases is described in Section III. This is followed by conclusion and future works in Section IV.

II. AIS

AIS is a data exchange system between ships that became mandatory by IMO (International Maritime Organization) in 2004. The AIS manages the sending and receiving of different information towards and from the surrounding vessels such as their GPS position, speed, course, type, time and place of arrival. AIS transceivers automatically broadcast information using a VHF (Very High Frequency) transmitter. This information is received by other ships or land based equipment. There are 27 different types of AIS messages that can be sent by AIS transceivers. It has had some advantages for maritime transportation like improvements in safety (collision avoidance, search and rescue) and management of fleets and navigation.

All AIS messages are coded and transmit three basic information at least: 1) The vessel's MMSI (Maritime Mobile Service Identity) which is a unique identification number, 2) The type of message being transmitted, and 3) A repeat indicator designed for repeating messages in case of issues with relay devices. An example of an AIS message is like:

!AIVDM,1,1,,B,35Mwq80000o9NfTJKjpAv2D62>`<,0*46

III. CIRMAR

CIRMAR is a scientific project for studying maritime traffic using AIS messages. It affects 1) geo-economy by integrating the maritime traffic with global logistics chains, 2) navigation and safety of maritime traffic, and 3) environmental impacts of maritime transportation. The project is a collaboration of some teams from different fields like geography, logistics and computer science. It is made of some different phases (cf. Figure 1).

A. Data acquisition

During the first phase, raw AIS messages are received from different sources which are 1) A local antenna installed at University of Le Havre, 2) AISHub data community, and 3) AIS data of GPMH[1] (Grand Port Maritime du Havre). The AISHub is a raw AIS data sharing service. It has 472 stations, which are geographically distributed all around the world tracking 21536 vessels[2]. It is necessary to share your local data with AISHub to access its data. That is why a local antenna is installed and sends the local AIS messages to the AISHub. The last source is GPMH, which is a state agency in charge of the port of Le Havre. The GPMH is a public institution of trade and industry and is responsible for management of the port and its facilities. We store all the messages received from these three sources every day in a text file with their exact arrival date and time.

[1] https://en.wikipedia.org/wiki/Port_of_Le_Havre

[2] http://www.aishub.net/, Date of access: 17 March 2017, 04.05pm

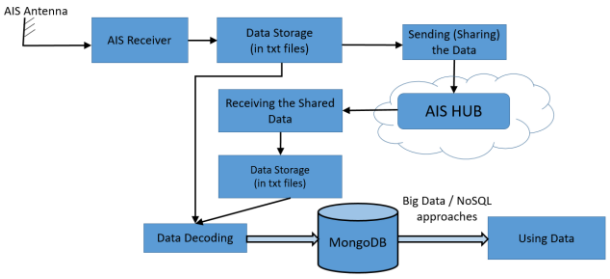


Figure 1. General Schema of the Project

B. Data processing

As the received messages are coded, a Java application using an open source library (AisLib[3]) is developed to decode the AIS messages. An important task that should be done during this stage is data cleaning. The data cleaning improves data quality by eliminating erroneous messages or correcting human-generated errors like port names. The correctness of a checksum included in each message should be checked and as a result, we ignore erroneous messages. In addition, only AIS message types 1, 2, 3, 4, 5, 18 and 19 are interesting for the CIRMAR project, so we ignore the other ones, too. The next step is to store AIS messages in the database.

C. Data storing

For this project, we have collected the AIS messages from the second half of 2015 and we have many files which are larger than 2GB, each of them including more than 10000000 AIS messages. This is why we consider it as the big data. And in fact, the management of these very large files is one of our challenges.

We have developed an application, which reads the collected files, and stores decoded AIS messages in our MongoDB[4] database. The MongoDB is a kind of NoSQL databases, which is suitable for big data applications. Also, it is very useful for unstructured data like the data we have in our project.

For a period of 6 months (second half of 2015), we have stored 1597337115 AIS messages, which occupy 169.18 GB. Table I shows the number of messages stored for each AIS message type.

In addition to the AIS messages, we have a database of different kind of ships (e.g. container, tanker, fishing, general cargo, liquefied gas and chemical ships). It includes a unique identification of all the ships, their name, registered owner (operator) country and company and also some other details like their deadweight, draught, length and built year.

The combination of these two databases makes it possible to join the data and analyze them. It is interesting to monitor the ships, the ports they passed, and the share of ships and their operators in the maritime traffic and commodity exchanges.

[3] https://github.com/dma-ais/AisLib

[4] https://www.mongodb.com/

TABLE I. AIS MESSAGES STORED DURING THE LAST 6 MONTHS OF 2015

Message Types	Number of messages
1 (Position report)	936618408
2 (Position report)	21497684
3 (Position report)	205201921
4 (Base station report)	52213304
5 (Static and voyage related data)	233886426
18 (Standard equipment position report)	140661167
19 (Extended equipment position report)	7258205

D. Interface

We have also made a web-based user interface using Ruby on Rails web development framework to let partners and users access and work with the stored data. At the moment, some basic reports and analysis are available. For example, it is possible to view all the messages received by a specific ship and explore the details. Also, it is possible to compare two months to understand the number of received messages, their types and the unique number of ships.

We are still working to make more interesting reports and analyzing tools available, for example to query the data based on a geographical region or relate the data to another database we are making to store the ships' companies and information. In the future, we will add some other big data technologies to facilitate working with the data and provide analyzing and visualization tools.

IV. CONCLUSION AND FUTURE WORKS

The goal of CIRMAR as a research project is to develop a proper platform to study maritime traffic. This aim is achieved by storing and analyzing AIS messages. The huge number of data received everyday makes it as a big data project with various challenges such as handling, storing and analyzing many large data files. In the other side, it has some potential applications such as:

- The punctuality of ships in ports in relation to the schedules of shipping companies.
- The impact on logistics chains of terrestrial links.
- The modeling of traffic conditions in busy seas (i.e. La Manche).
- The design (choice of ports of call) of regular shipping lines and their evolution.

For the future, we would like to add new features and improve the interface based on the requirements of users and new applications of the platform. In addition, it would be interesting to utilize more big data related technologies to improve and facilitate data analysis and visualization.

Scalable Guaranteed-Bandwidth Multicast Service in Software Defined ISP networks

Hardik Soni, Walid Dabbous, Thierry Turletti
Université Côte d'Azur, Inria, France
Email: *firstname.lastname*@inria.fr

Hitoshi Asaeda
NICT, Japan
Email: asaeda@nict.go.jp

***Abstract*—New applications where anyone can broadcast video are becoming very popular on smartphones. With the advent of high definition video, ISP providers may take the opportunity to propose new high quality broadcast services to their clients. Because of its centralized control plane, Software Defined Networking (SDN) seems an ideal way to deploy such a service in a flexible and bandwidth-efficient way. But deploying large scale multicast services on SDN requires smart group membership management and a bandwidth reservation mechanism to support QoS guarantees that should neither waste bandwidth nor impact too severely best effort traffic. In this paper, we propose a Network Function Virtualization based solution for Software Defined ISP networks to implement scalable multicast group management. Then, we propose the Lazy Load balancing Multicast (L2BM) routing algorithm for sharing the network capacity in a friendly way between guaranteed-bandwidth multicast traffic and best-effort traffic. Our implementation of the framework made on Floodlight controllers and Open vSwitches is used to study the performance of L2BM.**

I. Introduction

The massive increase of live video traffic on the Internet and the advent of Ultra High Definition (UHD) videos will require ISP providers to offer new services to support upcoming high-quality applications to their customers using multicast. Software Defined Networking (SDN) appears to be an appealing approach to implement and deploy innovative multicast routing algorithms in ISP networks [1–7] thanks to its logically centralized control plane. In addition, ISPs could exploit fine-grained control over QoS guaranteed multicast and best-effort traffic to implement traffic engineering policies that are friendly to low priority best-effort traffic. These approaches require costly real time monitoring of link utilization in order to allow network resources sharing between the QoS guaranteed and best effort traffic classes according to ISP traffic management policies.

Moreover, SDN-based centralized architectures suffer from well-known scalability issues. Different approaches either based on distributed [8–10] and hierarchical [11–13] control planes or on stateful data planes [14, 15] have been proposed to address SDN scalability issues in general. In the presence of large scale multicast applications, extra processing is required at routers to handle locally all Internet Group Management Protocol (IGMP) membership messages that would otherwise be flooded to the controller. In this work, we address these two problems: (1) how to avoid implosion of IGMP group membership messages at the SDN controller and (2) how to deploy guaranteed-bandwidth multicast services in Software Defined ISP networks with low cost and while being friendly with best effort traffic.

II. Scalable Multicast Group Management for Software Defined ISP Networks

To address the first problem, we propose to exploit the hierarchical structure of ISP networks and to use Network Function Virtualization (NFV). We delegate when needed the multicast group membership management through specific network functions running at the edge of the network as show in Figure 1. In our approach, NCs delegate multicast

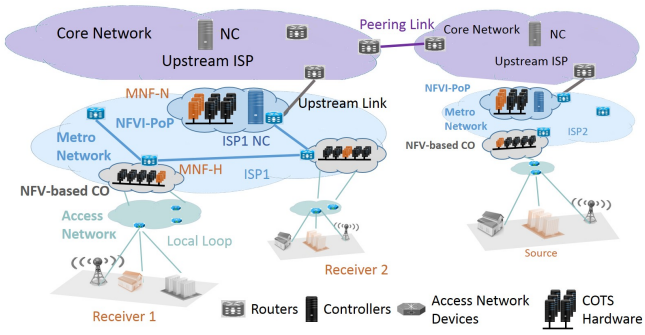
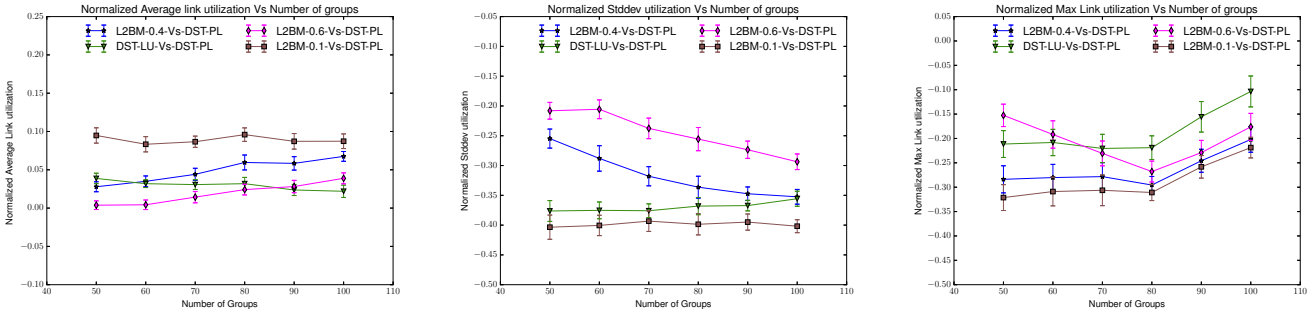


Fig. 1: Example of Software Defined ISP Network with NFVI-PoPs

group management functionalities to virtual network functions (VNFs) running at NFV Infrastructure at the edge of the metro networks. We call these functions MNFs for Multicast Network Functions, and define MNFs-H and MNFs-N. MNFs-H running in NFV-based COs exchange IGMP query and report messages with the downstream receivers. MNFs-N running in NFVI-PoPs process PIM Join/Prune signals sent by MNFs-H based on the membership states of their NFV-based CO. Unlike PIM Join/Prune messages, these signals do not update trees by themselves, but instead inform the corresponding MNFs-N to coordinate multicast tree update with the NC.

III. Lazy Load-balancing Multicast routing Algorithm (L2BM)

We propose a novel threshold-based load balancing algorithm in which a certain amount of link capacity, referred as the *threshold*, in the ISP's infrastructure is reserved in priority for guaranteed-bandwidth traffic. In absence of guaranteed-bandwidth traffic, best-effort can use this capacity. We dynamically increase the capacity share by gradually increasing the threshold. This approach is friendly to best-effort and



(a) Normalized Average link utilization (b) Normalized Standard Deviation of link utilization (c) Normalized Max link utilization

Fig. 2: Performance comparison of L2BM with different θ_{init}, DST-PL and DST-LU

helps in indirectly load-balancing the guaranteed-bandwidth traffic without the need of real-time link traffic monitoring mechanisms, as the controller is responsible of accepting or rejecting multicast subscription requests and is aware of bandwidth requirements. We use in forwarding devices Hierarchical Token Bucket [16], which is a classful queuing discipline that allows sharing the link capacity with flows of different priorities. Figure 3 shows a network with 4 nodes and a

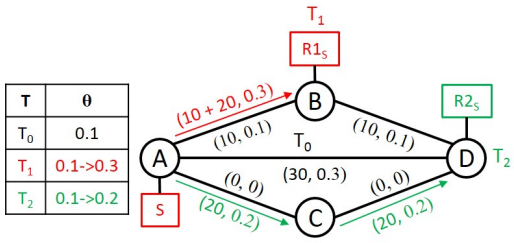


Fig. 3: Example of L2BM functioning

sequence of 3 events marked with numbers from T_0 to T_2. In this example, all links have the same capacity of 100 units. The current load and percentage link utilization are shown for each link e_{vu} using the notation (C_{vu}, U_{vu}). T_0 (in black) corresponds to the initial network state with existing multicast traffic. At T_1 (in red), receiver R_1 from node B joins a new multicast group with source S at A, which requires 20 units of bandwidth. As the link utilization of both e_{AB} and e_{DB} is equal to 0.1, the algorithm will explore both the edges, but will select tree node A increasing U_{AB} up to 0.3. At T_2 (in green), receiver R_2 from node D joins the same multicast group with source S. Again L2BM starts with $\theta = 0.1$, but ends up selecting the path A-C-D.

IV. Evaluation

We compare L2BM with two multicast routing algorithms that implement greedy heuristics of the Dynamic Steiner Tree (DST) algorithm with two different metrics: path-length (DST-PL) and link utilization (DST-LU) [17]. We virtualized *INTERNET2-AL2S* [18] network topology using DiG [19] on the Grid5000[1] network. We generate multicast groups with equal bandwidth demand of 2Mbps and associate 1 sender and 10 receivers per multicast group and the location of senders and receivers is randomly chosen across the set of nodes in the network. For every workload run, inter-arrival time of senders of different groups and receivers of the group are exponentially distributed with mean of $3s$ and $5s$, respectively. We make 20 runs of the workload with the same number of multicast groups and compute normalized Avg, StdDev and Max link utilization obtained with reference to DST-PL to compare the algorithms against DST-PL. 1) Average (Avg) refers to the overall network bandwidth consumed, 2) Standard Deviation (StdDev) estimates imbalance of traffic spread across the links and 3) Maximum (Max) corresponds to the most congested link in the network. These metrics are normalized for each run of the workload and averages of them are computed with 90% of confidence intervals.

DST-LU decreases the sum of links' utilization along the path, which increases with path length. On the other hand, L2BM minimizes the maximum link utilization using the threshold mechanism. Hence, it may use longer paths to reach any node in the multicast tree. The normalized Avg graph (Figure 2a) shows that L2BM has 5% higher Avg link utilization compared to DST-LU, except for the higher value of $\theta_{init} = 0.6$. However, both L2BM and DST-LU result in lower StdDev values compared to DST-PL as shown in Figure 2b, specifically for lower values of θ_{init}. It decreases the Max link utilization by 20 to 30% and 10 to 15% when compared against DST-PL and DST-LU, respectively, depending on the load and on the value of θ_{init}, see Figure 2c. To sum up, although L2BM consumes marginally higher link bandwidth than DST-LU, it succeeds to fairly share the overall network capacity between the guaranteed-bandwidth multicast traffic and the best-effort flows.

V. Conclusion

In this work[2], we propose a novel threshold-based load balancing algorithm to deploy at low cost a guaranteed-bandwidth multicast service that nicely cohabits with best effort traffic. By distributing the group membership management processing with NFs deployed at the edge of the infrastructure, we show that the NFV approach can be efficiently used to overcome the scalability issue of centralized SDN architectures.

[1]Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

[2]This work has been partly supported by Inria and the Japanese Society for the Promotion of Science (JSPS) in the context of the UHD-on-5G associated team and by the French ANR under the "ANR-13-INFR-013" DISCO project.

REFERENCES

[1] Y. Kok-Kiong et al. Towards software-friendly networks. In *ACM APSys*, pages 49–54, New York, NY, USA, 2010.

[2] C.A.C. Marcondes et al. Castflow: Clean-slate multicast approach using in-advance path processing in programmable networks. In *IEEE ISCC*, July 2012.

[3] L. Bondan et al. Multiflow: Multicast clean-slate with anticipated route calculation on openflow programmable networks. *JACR*, 2(2):68–74, 2013.

[4] S. Tang et al. Realizing video streaming multicast over SDN networks. In *CHINACOM*, pages 90–95, Aug 2014.

[5] A. Craig et al. Load balancing for multicast traffic in SDN using real-time link cost modification. In *IEEE ICC*, pages 5789–5795, June 2015.

[6] S. Q. Zhang et al. Network Function Virtualization enabled multicast routing on SDN. In *IEEE ICC*, 2015.

[7] J. Ruckert et al. Flexible, Efficient, and Scalable Software-Defined OTT Multicast for ISP Environments with DynSDM. *IEEE TNSM*, 2016.

[8] Y. Soheil et al. Beehive: Simple Distributed Programming in Software-Defined Networks. In *ACM SOSR*, 2016.

[9] K. Phemius et al. DISCO: Distributed multi-domain SDN controllers. In *NOMS*, pages 1–4, May 2014.

[10] K. Teemu et al. Onix: A Distributed Control Platform for Large-scale Production Networks. In *ACM OSDI*, 2010.

[11] H. Yeganeh et al. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In *ACM HotSDN*, 2012.

[12] M.A.S. Santos et al. Decentralizing SDN's Control Plane. In *IEEE LCN*, September 2014.

[13] Y. Fu et al. A hybrid hierarchical control plane for flow-based large-scale software-defined networks. *IEEE TNSM*, 12(2):117–131, June 2015.

[14] G. Bianchi et al. OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch. *SIGCOMM CCR*, 44(2):44–51, April 2014.

[15] H. Song. Protocol-oblivious Forwarding: Unleash the Power of SDN Through a Future-proof Forwarding Plane. In *ACM HotSDN*, 2013.

[16] HTB - Hierarchy Token Bucket, . https://linux.die.net/man/8/tc-htb. [Online; accessed 27-October-2016].

[17] B. M. Waxman. Routing of multipoint connections. *IEEE JSAC*, 6(9):1617–1622, Dec 1988.

[18] Internet2 AL2S Topology. https://noc.net.internet2.edu/i2network/advanced-layer-2-service.html.

[19] H. Soni et al. DiG: Data-centers in the Grid. In *IEEE NFV-SDN*, pages 4–6, Nov 2015.