Telemetry-Driven Optical 5G Serverless Architecture for Latency-Sensitive Edge Computing

István Pelle[1], **Francesco Paolucci**[2], **Balázs Sonkoly**[1], **Filippo Cugini**[3]

(1) MTA-BME Network Softwarization Research Group, Budapest, Hungary,
(2) Scuola Superiore Sant'Anna, Pisa, Italy, (3) CNIT, Pisa, Italy
pelle@tmit.bme.hu

Abstract: Latency-sensitive serverless subfunctions are optimally deployed at edge and cloud according to telemetry-retrieved data from the 5G transport infrastructure. Once deployed, serverless functions provided extremely fast invocation time of less than 450ms.
OCIS codes: 060.0060, 060.4250.

1. Introduction

Edge computing is driving the evolution of optical 5G converged fronthaul and backhaul network architecture towards a unified computing and network infrastructure able to fulfill latency-sensitive requirements of emerging 5G services. Latency-sensitive data have to be elaborated in close proximity to where they are produced, making inadequate the traditional approach of fully delegating all computations to remote cloud resources. However, the availability of computing and networking resources at the edge is limited for cost reasons. Moreover, such resources have to be shared among multiple competing and dynamic 5G applications. An example of resource-hungry 5G applications is represented by Artificial Intelligence (AI) inferences, such as video analytics, offloaded from end terminals but often requiring near-real time capabilities in object/event recognition.

Recent advances at the optical 5G network infrastructure level have enabled attractive features in terms of data plane programmability and monitoring capabilities. E.g., the P4 technology has emerged as one of the most relevant candidates to provide network programmability over optical 5G infrastructures. In [1], a P4-defined data path for L2/L3 transport for sliceable optical transport is demonstrated, while [2] showcases monitoring and capturing latency performance of applications by using in-band telemetry. In addition, telemetry has been exploited to monitor network resources with higher accuracy at limited bandwidth compared to traditional approaches [3].

To effectively support latency-sensitive 5G applications accounting for limited availability of edge resources, we propose to leverage on a comprehensive architecture exploiting *(i)* accurate network monitoring infrastructure and *(ii)* Function as a Service (FaaS, serverless) technology, here applied at the edge. In serverless, traditional monolithic applications are engineered as a combination of (sub)-functions to be deployed and combined/chained in a flexible way. So far, serverless has been implemented within data centers (e.g., Amazon Web Services) or in on-premises remote facilities, but to the best of our knowledge, it has never been experimented at the edge in a latency-sensitive optical 5G scenario provided with detailed and comprehensive monitoring capabilities.

2. Telemetry-driven 5G network architecture for latency-sensitive serverless computing

Fig. 1 shows the reference optical 5G edge-to-DC converged fronthaul and backhaul network architecture. It extensively exploits telemetry and data plane programmability to provide deep network awareness to serverless control systems. Telemetry-based monitoring is activated from all network elements (i.e., optical and packet systems). P4-enabled NICs are used in edge compute nodes, providing direct optical connectivity augmented with in-band monitoring capabilities. In particular, P4 programmability is exploited to introduce ad hoc header extensions directly by the NIC, thus accounting for latency contributions (e.g., queuing time) directly from the server and updated throughout the whole network up to the data center. The proposed architecture encompasses a serverless control system based on three main functional modules, shown in Fig. 2. The *Edge-Cloud Optimizer* (ECO) is responsible for determining the placement and layout of components of an application that is given by its functions (smallest individually deployable components, f_i) and the call hierarchy among those. Based on quality of service (QoS) indicators measured through telemetry on the 5G transport and computing infrastructure, the ECO computes an optimal grouping (SF_i) of the application's functions and assigns these to either edge or cloud resources while satisfying the application's QoS requirements. Actual deployment is performed by the *Serverless Deployment Engine* (SDE) component via calls through the *Provider API* that has direct connection to cloud and edge resources. After deployment, the *Telemetry Service* (TS) module monitors application and network performance and reports it to the ECO.

Fig. 1 also shows the use case of serverless function instantiation deployed at the edge and in the cloud. *Serverless Function 1* (SF_1) is instantiated close to the user equipment to perform low-latency operations (e.g., image processing and inference for critical object recognition), while SF_2 is deployed in the cloud performing offloading operations subject to bounded latency constraints (e.g., image upload to a repository). The Telemetry Service

operates on different transport segments, spanning different layers. The *Optical Telemetry* (OT) module collects Quality of Transmission (QoT) monitoring information from the optical layer leveraging disaggregated streaming of Optical Signal-To-Noise Ratio (OSNR) values at the coherent receiver and power values at intermediate nodes [3]. The *Network Telemetry* (NT) module collects packet statistics of target traffic in the form of in-band telemetry mirrors, exploiting P4 switches. The *Edge Telemetry module* (ET) collects IT statistics, such as edge CPU rate and memory usage. This way, all layers are monitored to detect QoS degradation. For example, in a case where N_{J2} is affected by extra traffic that introduces additional latency, or an optical connection experiences a transmission degradation, a SF_1 migration to node E_2 can yield better overall application performance.

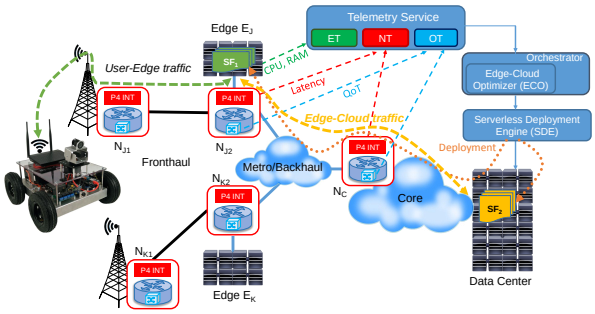

Fig. 1: 5G and Edge telemetry: architecture.

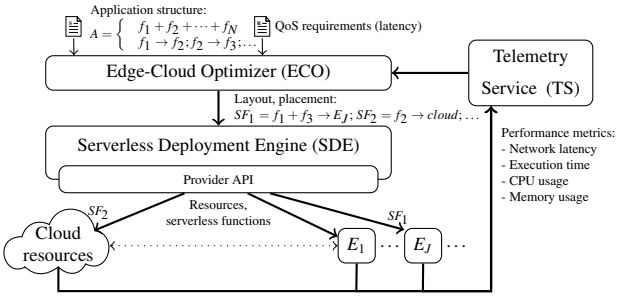

Fig. 2: Schematics of code deployment and monitoring.

3. Telemetry and Serverless Deployment Engine Workflow

The proposed architecture has been implemented with specific focus on two main modules: the Telemetry Service, expanded upon previous works for comprehensive telemetry management, and the new Serverless Deployment Engine. The TS resorts to telemetry points in the 5G transport network enabling fast detection of optical impairments and queue congestion on packet nodes due to dynamic traffic conditions (e.g., a heavy load of edge SF deployment connected to the same backhaul node). To this goal, the SDN P4 switches perform in-band telemetry (INT). Excerpts of P4 code are shown in Fig. 3. Each P4 switch, in the ingress section, defines and applies an extra header of type INT storing latency information computed by the P4 node itself (action *add_int*). Moreover, dynamic cloning of selected service traffic headers is activated to provide INT header to the TS (action *do_mirror*). At the egress section, the switch computes the queue time metadata and writes it into the header (action *write_int*) and, for mirrored traffic only, it truncates the packet to provide just the header stack (action *do_truncate*). This is repeated for all the traversed nodes, until the last domain node (e.g., N_C) extracts INT and removes extra headers, thus avoiding the telemetry to reach the end user and the cloud. The TS NT module collects per-packet latency and builds statistics to be provided to the ECO.

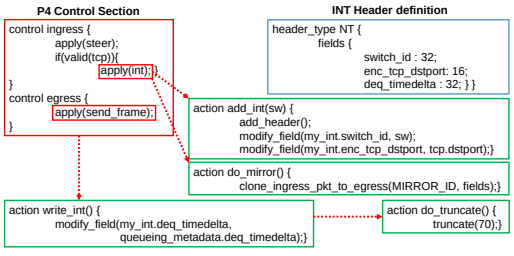

Fig. 3: P4 code employing telemetry system.

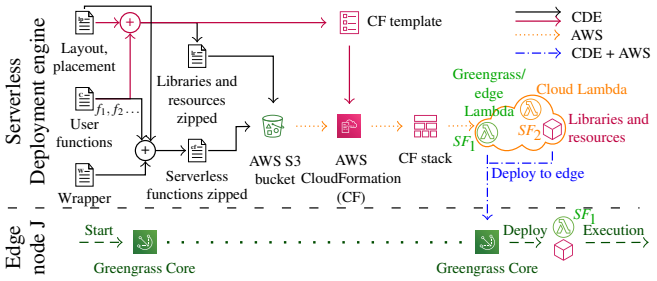

Fig. 4: Code deployment stages.

The proposed implementation of the SDE first receives layout and placement descriptions from the ECO and based on those, assembles groups of serverless functions complimented by *wrapper* code. Then, it adapts user code to different (edge or cloud) environments, collects additional libraries and file resources (e.g., machine learning artifacts), and determines their placement and accessibility according to ECO specifications. At the compute level, the SDE relies on the solutions offered by AWS for deploying application components. First, components are zipped and uploaded to the cloud AWS's object storage service (called S3). The SDE assembles an AWS CloudFormation (CF) template that contains the placement, resource flavor and connection information of all the required cloud resources. The template is then initialized in the cloud (relaying on AWS CloudFormation deployment engine). At the same time, the SDE translates layout and placement specifications to specific cloud and edge compute and database resource definitions. Serverless functions are then deployed in the cloud using AWS Lambda technology. Subsequently, the SDE deploys serverless functions at the edge relying on the AWS Greengrass technology. Finally, Greengrass service initializes the serverless code on the edge node.

4. Experimental Results

The proposed telemetry-driven serverless architecture has been implemented and validated in an edge-cloud network testbed employing packet-over-optical metro networks including P4 switches and optical transport through transponders and ROADMs. The testbed is shown in Fig. 5 where the edge node is a Linux server equipped with four Intel Xeon E5-2650 v3 vCPUs, 6 GB of memory, Ubuntu 18.04, P4 NICs with optical interfaces, and a wireless interface providing wireless connection to the rover. The considered latency-sensitive application, implemented as set of serverless functions, performs remote driving of a rover based on detected objects provided by a video camera on the rover to the edge node. Querying an image from the rover's camera and sending control messages to the rover is always performed at the edge device while object detection can be deployed either on the edge or in the cloud based on latency performance. The latter would be preferable to preserve edge resources, but it can be implemented only if limited extra latency is introduced by the network. P4 switches are implemented over Linux server boxes equipped with 10 and 40 Gb/s optical interfaces and implementing the Behavioral Model version 2 (BMV2) with P4 code supporting custom INT and selected mirroring of packet headers.The switches are connected to the optical layer by means of commercial 100G optical cards and QoT telemetry is realized through NETCONF-based subscription to gRPC streaming of OSNR values received at the card (e.g., as in [3]). The TS and SDE modules are implemented in Python. Additional dynamic traffic is injected in the backhaul network using a Spirent N4U generator, connected with the Edge P4 Switch through 10G optical interfaces.

Fig. 6 (left) shows the latency detected by the TS system and experienced by the traffic flowing between the edge node and the AWS cloud due to a single backhaul node with different traffic scenarios. To measure significant network latencies, the P4 queue transmission rate has been limited to emulate real network congestion. Results show that without synthetic traffic generation (NO) the latency is always below 0.1ms. If the optical path is longer or the external traffic increases, latency may reach warning conditions. For example, it remains in the 0.15–0.3ms range when queues are not full, under the traffic threshold (UT). Further congestion may occur, e.g. in case of failure in the optical network, leading to additional latency. Above the threshold (AT), traffic experiences significant delays in the node queue, in the range of up to 12ms, impacting on edge-cloud QoS. As shown by Fig. 6 (center) the object detection executes 250ms quicker on a high capacity AWS server than on our edge device. However, AWS calls add significant latency to cloud execution (at least 250ms [4]). As the ECO is aware of these conditions, with having an uncongested backhaul network, deploying the object detection to the cloud might yield better results. Changing network conditions, however might force the migration of the function to the edge node. In such a case, redeployment with the SDE takes 113s: 55s for updating resources in the cloud using CloudFormation and another 58s for edge deployment. This is performed under warning latency conditions (e.g., UT). After code is loaded to the edge, the Greengrass Core starts new function instances. Fig. 6 (right) shows a comparison between the startup time of Serverless Greengrass and Docker containers which indicates that, on average, the former takes 1154ms for executing code on the edge compared to the latter which requires 446ms.

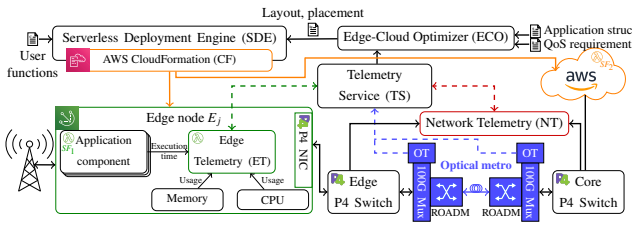

Fig. 5: Testbed setup and components of the telemetry system.

P4 Tx Rate [kframe/s]	Threshold [Mb/s]	P4 Node Latency [μs] NO	UT	AT
5	10	<100	200	12800
10	110	<100	300	6300
20	120	<100	120	3170
40	160	<100	150	1540

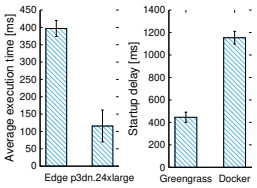

Fig. 6: Performance metrics: network (left), object detection (center) and container startup (right).

5. Conclusions

A latency-sensitive rover driving application was decomposed as serverless subfunctions optimally deployed at either the edge or the cloud according to latency and congestion telemetry-retrieved data from the optical 5G transport network. Deployed serverless functions also showed setup time of 446ms, much lower than traditional Docker-based solutions experiencing more than 1s.

References

1. Y. Yan *et al.*, "P4-enabled smart NIC: Architecture and technology enabling sliceable optical DCS," in *European Conference on Optical Communications (ECOC)*, 2019, pp. 1–3.
2. F. Cugini *et al.*, "P4 in-band telemetry (INT) for latency-aware VNF in metro networks," in *OFC*, 2019, p. M3Z.6.
3. F. Paolucci, *et al.*, "Network telemetry streaming services in SDN-based disaggregated optical networks," *IEEE JLT*, vol. 36, no. 15, pp. 3142–3149, Aug 2018.
4. I. Pelle *et al.*, "Towards Latency Sensitive Cloud Native Applications: A Performance Study on AWS," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, July 2019, pp. 272–280.