5G-Slicer: An emulator for mobile IoT applications deployed over 5G network slices

Moysis Symeonides*, Demetris Trihinas†, George Pallis*, Marios D. Dikaiakos*,
Constantinos Psomas‡, Ioannis Krikidis‡

* Department of Computer Science
University of Cyprus
{msymeo03, gpallis, mdd}@cs.ucy.ac.cy

† Department of Computer Science
University of Nicosia
trihinas.d@unic.ac.cy

‡ Department of Electrical and
Computer Engineering
University of Cyprus
{psomas, krikidis}@ucy.ac.cy

***Abstract*—5G is emerging as a key mobile network technology offering Gbps transmission rates, lower communication latency, and support for 10-100x more connected devices. The full exploitation of 5G relies on network slicing, a network virtualization technique where operators split a physical network among a wide number and variety of services, in accordance to their individual needs. However, experimentation with 5G-enabled services and measurement of key performance indicators (KPIs) over network slices is extremely challenging as it requires the deployment and coordination of numerous physical devices, including edge and cloud resources. In this paper, we introduce 5G-Slicer; an open and extensible framework for modeling and rapid experimentation of 5G-enabled services via a scalable network slicing emulator. Through modeling abstractions, our solution eases the definition of 5G network slices, virtual and physical fog resources, and the mobility of involved entities. With the blueprint of an emulated testbed at hand, users can create reproducible experiments to evaluate application functionality and KPIs by injecting load, faults and even changing runtime configurations. To show the wide applicability of 5G-Slicer, we introduce a proof-of-concept use-case that encompasses different scenarios for capacity management in a city-scale intelligent transportation service. Evaluation results exploiting real 5G data show that 5G-slicer presents, at most, an 11.7% deviation when comparing actual and emulated network Quality of Service (QoS).**

***Index Terms*—Network Slicing, Edge Computing, Mobility.**

I. Introduction

The proliferation of Internet of Things (IoT) technologies fosters a variety of novel application areas, like autonomous drone swarms, connected vehicles and digital twins for industrial IoT. These applications typically present demanding requirements, which require fast response times, and involve numerous, heterogeneous mobile sensors inter-connected and inter-weaved with both physical and virtual networks that create unified cyber-physical planes [1]. However, the complexity and cost of purchasing, deploying, and maintaining such complex network fabrics are key inhibitors to the development of delay-sensitive IoT applications [2]. Also, the rate at which data are produced at the network edge further prevents the adoption of such applications at a large-scale [3].

The 5th generation (5G) of mobile communication promises to satisfy the pressing requirements of emerging IoT applications by offering a 1000x higher mobile data volume per unit area and support for 10–100x more connected devices [4]. *Network Slicing* is a key innovation of 5G technology and has the potential to provide the building block for reducing the cost and complexity of managing large-scale IoT applications [5]. Network slicing facilitates the provision of multiple logical networks on top of a physical network for Operators to rent "slices" of their infrastructure, much similar to cloud providers renting computing resources. These "slices" can be tailored to address specific applications needs, through guarantees provided by the network Operator. However, ensuring QoS guarantees is hard, especially for applications with highly dynamic characteristics. In the case of connected vehicles, connectivity can change due to various factors, such as varying distance to the closest radio unit, the appearance of physical obstacles, and the choice of wireless protocol. In turn, compute-hungry tasks may be offloaded to more powerful Multi-access Edge Computing (MEC) servers during execution [6]. Finally, the sharing of the underlying network infrastructure can result in situations where slices on the same MEC compete for insufficient computing and communication resources with unexpected adverse effects on application behavior [7].

To cope with these challenges, service owners can either purchase expensive, high-coverage slices with dense access points and multiple MECs or design their slices to be capable of handling highly volatile mobile workloads while maximizing network coverage. However, undertaking such a design requires powerful tools that support extensive experimentation, testing, and performance evaluation. These tools should capture accurately the key features of the underlying infrastructure and the target application while supporting the easy configuration of scenarios, generation of appropriate workloads, monitoring of important metrics and KPIs. A common approach that seeks to address these requirements, entails the combined use of mobility and network simulators [8], [9]. This, however, involves a steep learning curve of multiple tools and complex configurations, which cannot be capitalized in the application development life-cycle. Also, developers face difficulties when trying to map the business logic of an IoT application to a simulator design model, because simulators do not capture all aspects of such complex infrastructures. So, for instance, a developer must extract quantitative metrics from a small-scale experiment and introduce these metrics in a simulator model

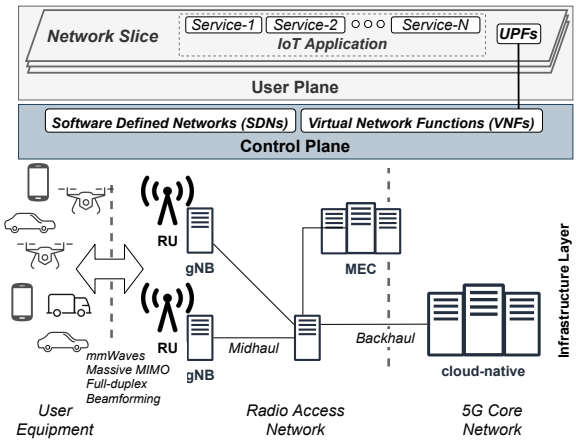

Fig. 1: A High-Level Architectural Overview of a 5G Network

to drive more realistic simulation scenarios [6].

Emulators try to address these issues by mimicking the actual environment where a real-world application is deployed and runs [10]. However, emulators that focus on 5G infrastructure [11] or on mobility patterns [12] fail to support adequately the concept of Network Slicing as: (i) they do not support key features like Virtual Network Functions (VNFs) and (ii) lack the means to scale the emulated execution environment to large-scale application scenarios. To address these limitations, we introduce 5G-Slicer, a framework that facilitates the emulation of Network Slicing in the presence of large-scale and mobile IoT applications. 5G-Slicer supports: (i) modeling abstractions for network infrastructure, such as radio units, MECs, mobile nodes, user equipment, and node trajectories; (ii) dynamic updates of radio network characteristics (i.e., signal strength) and respective network QoS, according to the mobility patterns of end-nodes; and (iii) primitives for describing user-plane VNFs, such as software firewalls, policy enforcement enablers, and packet inspection. Users simply describe the experimentation scenario, leaving to 5G-Slicer its transformation into an emulated testbed. In summary, the main contributions of this paper are:

- A **comprehensive model** that captures the key characteristics of a 5G network slice along with the mobility of network entities. The model's expressivity enables users to design and build complex 5G network slices, encapsulating QoS definition, user-plane network functions, physical components, such as access points and base stations, physical nodes' positioning and trajectories, new network technologies (multi-user MIMO and beamforming), and virtualized MEC and Cloud resources.
- An **open-source framework**[1] that takes as input the slice model and generates an experiment testbed, while also offering extensible interfaces for dynamically altering at runtime network QoS based on mobile node positioning. 5G-Slicer provides dynamic maps with real-time object location updates to help users extract insights, visually contrasting the changing locations of emulated nodes against real-time application performance metrics.
- A **comprehensive evaluation** of 5G-Slicer emulation accuracy after comparison to real measurements obtained from a 5G testbed for various settings and protocols. In all configurations the emulation error is at most 11.7%.
- A **proof-of-concept use-case**, where 5G-Slicer is used to explore a city-scale intelligent transportation service based on real-world data. The use-case is reproducible and highly configurable (e.g., density of access points, edge servers, node mobility). Based on this, we perform an extensive experimentation on many high-scale mobile edge topologies with different characteristics and retrieve insights for the performance of the topologies involved (e.g. for capacity planning).

The rest of the paper is structured as follows: Section II provides technological aspects for 5G networks. Section III introduces 5G-Slicer, with Section IV showcasing the modeling abstractions and Section V implementation details. A detailed evaluation is introduced in Section VI. Finally, Section VII presents related work and Section VIII concludes the paper.

II. Background

Compared to previous mobile network generations, 5G combines physical layer improvements with emerging network virtualization to enhance the connectivity and energy-efficiency of any "thing" that stands to benefit from being connected [13]. Figure 1 depicts a high-level overview of a 5G network. The physical infrastructure includes User Equipment (UE), Radio Access Network (RAN), MEC nodes, and a cloud-based Core Network. UE corresponds to any end-user device that interacts with the network, such as phones, wearables or embedded devices, connected vehicles, and even nodes with enhanced computing capabilities (e.g., self-hosted edge servers). UEs interact with the rest of the network through high-performance communication protocols such as multi-user massive MIMO. This increases system capacity and minimizes UE-RAN network delay by combining a large number of antennas along with the gains of increased bandwidth by operating on the millimeter wave (mmWave) band [14].

The RAN is comprised of wireless base stations (gNodeB or gNB) connecting UEs to the 5G core network. gNBs follow the 5G disaggregation model [15], and can be assembled into hierarchical structures, wherein each gNB may control one or more Radio Units (RUs). RUs include antennas and radio frequency circuits for voice and data transmission, along with processing capabilities. RUs embrace *beamforming*, the process used to produce narrow beams in the mmWave band that can be controlled by the antenna to point to specific directions. gNBs are connected with each other and with a nearby MEC through midhaul connection; their internal structure is not visible to the core network and other RAN nodes. This architecture aims at improving the network's energy efficiency by reducing centralized processing and consequently reducing the Network Operators' operational expenses [16]. The last infrastructure element is the cloud-native Network Core (NC), where Operators deploy resource-demanding virtualized network services. The communication between the RAN

[1] https://github.com/UCY-LINC-LAB/5G-Slicer

and cloud is realized via backhaul connections, which have significantly higher network latency than midhaul connections.

On top of physical infrastructure, a virtual layer is provisioned and separated into control and user planes. This separation enhances infrastructure management via the virtualization and isolation of the underlying resources. Network management relies on Network Function Virtualization (NFV) and Software Defined Networks (SDNs) to provide a set of standardized control plane functions and to decouple network functions from the hardware. Hence, network operators have the flexibility to (i) scale in/out services based on client demands; (ii) decrease capital and operational expenditure via virtualized components and ad-hoc infrastructures; (iii) minimize time-to-market for new services and (iv) sell virtualized end-to-end networks, the *slices*, to their customers [13].

With Network Slices (NS's), a physical network is split into multiple isolated logical networks of different size, structure, and functionality. Through NS, customers describe the application, network, and infrastructure via human-readable configurations. The Network Operator will strive to fulfill the desired QoS for each NS, as described in a service level agreement (SLA) [7]. Hence, the network infrastructure can be decomposed to address the requirements of different applications. For example, one slice can be dedicated to mobile broadband applications that require high throughput, while another slice can be dedicated to machine-to-machine communications (M2M) that require high connection density. A slice usually spans across all layers of a 5G network. In a nutshell, it can be seen as a Software-Defined Network along with a set of VNFs and user-defined services supported by MECs. Based on the 3GPP specification, network functions on a network slice are equivalent to User Plane Functions (UPFs) that can be programmed for packet inspection, routing, and traffic filtering [17]. To this end, the abstraction of NS provides an end-to-end multi-service orchestration plane, eases network programmability, and reduces the implementation efforts for both providers and clients.

III. The 5G-Slicer Framework

This Section provides a high-level overview of 5G-Slicer and elaborates on its functionality from a user perspective.

A. Requirements

In a typical example, a 5G-enabled application is deployed on a rented network slice that abstracts the underlying network infrastructure while adhering to well-defined SLAs. However, with thousands of mobile nodes connected over the RAN, developers cannot be sure if the given guarantees are sufficient or even overly cautious. Thus, the following requirements must be taken into consideration when designing an emulator for mobile IoT applications deployed over 5G network slices:

R1. Cloud, MEC and UE modeling: the ability to deploy IoT applications over virtualized resources on the Network Core, nearby MECs, and/or on physical UE with these resources featuring a diverse degree of heterogeneity.

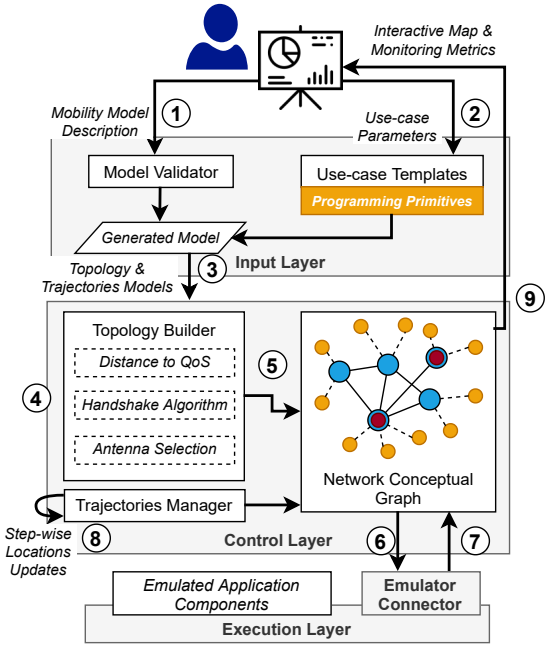

Fig. 2: A High-Level and Abstract Overview of 5G-Slicer

R2. Virtualized Networks: the ability to design on-demand network slices that are based on SDNs and are equipped with VNFs, such as firewalls, routing schemes and usage limits.

R3. Geo-Positioning & Mobility: the ability to apply at runtime ad-hoc and/or trajectory-based positioning changes to mobile nodes that may impact connectivity and performance.

R4. Testbed Scalability: the emulation framework should only be bounded by the resources reserved for the underlying execution environment and when the resource pool expands, so should the responsiveness of the emulator.

R5. Network Slice Monitoring: To assess the performance of network slicing, real-time quantitative network metrics are required, as well as packet-level inspection, resource observability, and even application behavior metrics.

B. Framework Overview

5G-Slicer is an emulation framework which seeks to enable 5G application developers to focus on the business logic of their services instead of the whole communication stack when trying to analyze application performance. Users of 5G-Slicer need only to define the testbed configuration and test scenarios via modeling abstractions, leaving to 5G-Slicer the burden of the deployment, configuration, and monitoring of the emulated execution environment of a 5G network slice. 5G-Slicer is capable of orchestrating run-time changes to the positioning of mobile nodes and, at the same time, updates the connectivity (e.g., signal strength, data rate) of impacted entities.

Figure 2 depicts a high-level overview of the 5G-Slicer components and their role in a typical use-case. The deployment of a typical experiment starts with a user describing the application services and the 5G network via the model specification ① or by parameterizing a "ready-to-use" testbed template ②. The description specifies capabilities of the virtual and physical infrastructure wherein the application's

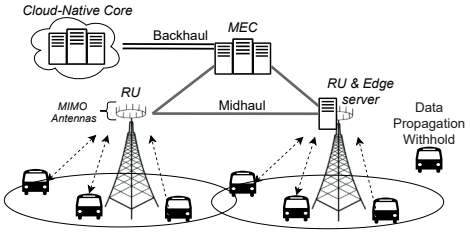

Fig. 3: Smart Bus Network Use-Case Scenario

services reside and interact with, such as UEs, edge and MEC servers, and cloud resources. The user can also define the geo-positioning of the infrastructure and provide fixed trajectory changes that will occur during the emulation or more comprehensively, annotate mobile nodes with a trajectory model that the node will adhere throughout the emulation. Then, the user can define the network slices that the network fabric will support. A network slice is abstracted as a virtual overlay mesh that runs on top of the network infrastructure. The user can also specify performance characteristics for the connections between RAN nodes and the backhaul, and the wireless connections between the RAN and UEs. For the latter, users can experiment with alternative degradation models, which specify how physical distance influences the connectivity between RUs and mobile UEs (e.g. linear), and/or with alternative wireless connection protocols (e.g. MIMO). Finally, the user can optionally add VNFs to nodes, such as firewalls, traffic filtering, packet inspection, etc. The modeling specification is described in detail in Section IV.

If a user is interested in evaluating the performance of a 5G environment with certain settings, 5G-Slicer provides a repository with easily configurable and "ready-to-use" templates. A template enables the rapid deployment of a 5G application with the infrastructure, entity mobility and monitoring already set. Users can configure the use-case via a set of parameterizable configurations, such as the number of entities (i.e., mobile nodes, MECs), their density, the type and connectivity QoS of the mobile networks, and the deployed services. To further facilitate the design of large-scale infrastructure descriptions, 5G-Slicer provides programming primitives that establish a programmable view of the model.

When the description is complete ③, the system submits the description to the Topology Builder. The role of the **Topology Builder** ④ is to assess the model validity and to compile it into a *Network Conceptual Graph*. To achieve this, the Topology Builder extracts from the given description the network slice specification (i.e., wireless protocol, QoS of RAN nodes) and any signal degradation models defined during the modeling process. With these, the deployment will invoke three sub-modules: (i) *Antenna Selection*: this evaluates if a mobile node can reach a network and selects the RUs to connect; (ii) *Distance-to-QoS process*: this assigns network QoS to the respective links; and (iii) *Handshake Algorithm*, which introduces additional delay when a moving node connects to new RUs. The output of this process is a **Network Conceptual Graph** ⑤ that contains the aforementioned information and

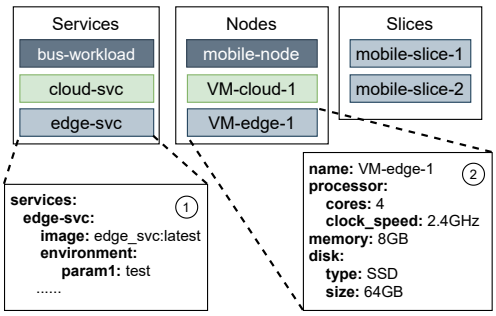

Fig. 4: Building blocks of Smart Bus Network Scenario

will be used by the system for the runtime state propagation during the experimentation. The nodes of the graph that represent network and compute devices are annotated with information about their capabilities and deployed services. Edges denote the links between the nodes. The weight of each edge is determined by a set of network QoS parameters, such as data rate, network delay, packet error rate, etc.

With the conceptual graph in hand, 5G-Slicer proceeds with the construction of the emulated execution environment. Specifically, 5G-Slicer invokes the **Emulator Connector** ⑥ that interconnects the conceptual graph with the underlying emulation environment. Its responsibilities include spawning the emulated instances, configuring the network QoS parameters between them, enforcement of any run-time actions and mobile node trajectory updates. Through the Emulator Connector, monitoring data is extracted from the execution environment and the conceptual graph is updated with new state context ⑦. All ad-hoc changes to the geo-positioning of mobile nodes is handled by the **Trajectory Manager** ⑧. Hence, the role of the Trajectory Manager, is to orchestrate "in-time" location updates for mobile nodes. When a location update occurs, the Trajectory Manager populates the change to the conceptual graph, and, consequently, the system automatically propagates any changes to the execution environment.

The final step of the process is the performance analysis and visualization of the emulation results. The framework facilitates this by interacting with the conceptual graph and generating an **Interactive Map** along with a set of representative visuals from the **Monitoring Metrics** ⑨. A user can then extract useful analytic insights for capacity and cost management, network KPI assessment, SLA violations, etc.

IV. Model Description

The 5G-Slicer model introduces an end-to-end network slicing description along with positioning and mobility primitives. Specifically, the model is composed of: (i) *Services*, facilitating the description of 5G-enabled services; (ii) *Nodes*, introducing virtual and physical compute resources; (iii) *Network Slices*, indicating network infrastructure characteristics; (iv) the *Deployment*, specifying the placement of the services, and physical positioning; and (v) *Move Actions & Trajectories*, which denote runtime alterations of UE positioning.

To ease the understanding of the model concepts, let us consider an exemplary use-case of a Bus Operator that equips

its fleet with IoT devices for real-time tracking, in an attempt to optimize the quality of offered services. In Figure 3, IoTs gather data, such as bus location, operating area, route delay, and periodically propagate collected data through the radio network of a purchased 5G network slice. If an RU is in the vicinity of the bus, then data is propagated through it to the nearest MEC for pre-processing, otherwise the data is stored on IoT memory until it reconnects to the network. After data pre-processing, the data is propagated to the Bus Operator's cloud for persistent storage and offline data analysis.

A. Services & Nodes

A *Service* in 5G-Slicer is a containerized, independently deployable, and scalable software component that represents application business logic on top of a 5G network slice. Furthermore, a *Service* can be an IoT data generator that the developer utilizes to stress the deployed application (e.g., the bus workload service in Figure 4 ①). We adopt containers as the key abstraction for defining and deploying application workloads on 5G-Slicer, because of their power, simplicity, and wide use with popular cloud application patterns such as microservices [10]. 5G-Slicer Services are inherited from the docker-compose specification,to ease the deployment description of application services. On the other hand, a *Node* can be seen as a constrained compute entity of the emulated execution environment and can be described with attributes such as the node name, identifier, etc. Most importantly, a node can be annotated with processing capabilities, including, processor power, memory, and disk capacity. For instance, the VM with name *VM-edge-1* (Figure 4 ②) is equipped with 4 cores@2.4GHz, 8GB RAM, and 64GB SSD disk.

B. Network Slice & Connection Types

A network slice (Figure 5 ①) is described by: (i) name (and identifier); (ii) midhaul QoS, connectivity between RUs and Edge; (iii) backhaul QoS, connectivity between RUs and Edge to the 5G-enabled services; (iv) the wireless connection type, which specifies the technological aspects of the connectivity between the UE and RU; (v) a set of VNFs; and (vi) a set of RUs. The midhaul and backhaul connections, unless otherwise stated, have stable and guaranteed QoS, including data rate, network delay and packet error rate.

A slice description is annotated with VNFs, which are equivalent with user plane functions in a 5G slice. In Figure 5 ①, *mobile-slice-1* features a set of VNFs. In this, 5G-Slicer will capture measurements for in- and out-going packets using the *packet-monitoring* property set. Furthermore, our model provides users with the ability to apply firewall rules, such as packet drop or redirect, which can be specified on protocols, ports, other emulated nodes, or even external IPs. Hence, *mobile-slice-1* includes a rule to *DROP* all *TCP* packets coming from the *UE-workload* service to port *8080*. Finally, in a network slice, users can form their own traffic policies to reduce overheads or to manipulate the service performance. In Figure 5 ①, the *traffic_policy* primitive is used to limit the data rate of UDP packets in the slice to $100Mbps$.

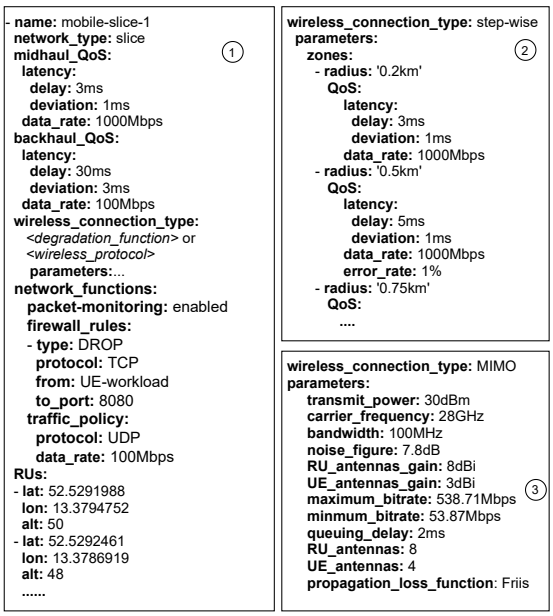

Fig. 5: Exemplary Network Slice & Connection Descriptions

Contrary to midhaul and backhaul, the QoS of wireless connections is not fixed. The distance between the transmitter and receiver is a dynamic parameter influencing signal strength and consequently the QoS of a wireless channel. To support distance-based computation of QoS, 5G-Slicer supports mathematical degradation models, i.e., linear, logarithmic, and step-wise, that users can parameterize with slight effort. Figure 5 ② depicts a step-wise approach, where the user defines 3 stages: (i) 0 to $200m$, the network delay is set to $3ms$ and the data rate $1000Mbps$; (ii) 200 to $500m$, these parameters are now $5ms$ and $1000Mbps$, with an error rate of 1%; and (iii) 500 to $750m$, the connectivity is characterized by $10ms$ delay, $750Mbps$ data rate, and 2% error rate. Each mathematical model takes different parameters, which can be set by the user. For instance, for the linear and logarithmic formula the user can define the maximum distance (radius) a mobile node can be reached by a RU, the best and worst QoS attained, etc., whereas for the static model the user needs to define flat QoS values attained when the UE is in the RU's reach.

Alternatively to the degradation models, 5G-Slicer supports the enablement of realistic wireless physical effects for well-known 5G protocols like multi-user MIMO and Beamforming. Figure 5 ③ depicts the configuration of a RAN adopting multi-user MIMO for its RUs. This simply requires the annotation of the description with the RU transmission power, frequency, UE and RU antennas gains, max and min bit rates, propagation loss function, and antenna elements for both UE and RU. At run-time, the system uses the parameters and "translates" the UE-RU distance to respective network QoS.

C. Deployment Description

Figure 6 introduces the *Deployment* primitive that realizes the placement of the emulated 5G-enabled services on top of one or more network slices. A Deployment is a set of *Blueprints* that are described by the user to realize the service

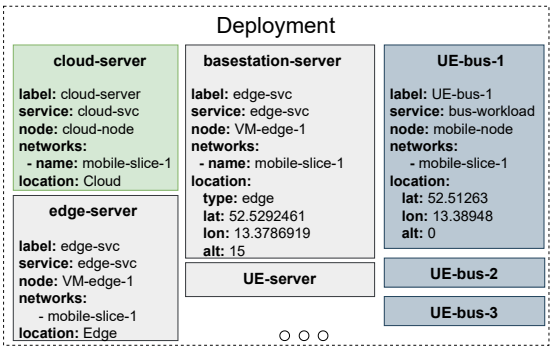

Fig. 6: Exemplary Deployment Description

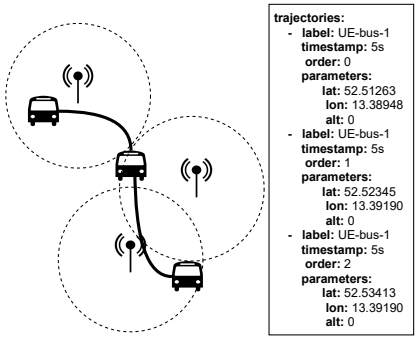

Fig. 7: Mobile Node Description Annotated with Trajectory

placement on the network slice. A Blueprint combines: (i) a *Node* that will execute the service, (ii) the respective *Service* that is defined in a docker-compose file, (iii) a set of *Networks*, which denote the different slices that a node is connected to, (iv) a *label* that works as unique identifier for the system, and (vi) the *location* primitive. The location can be either the physical geographic location of a node or a virtual location. The geographic position of the node contains the latitude (*lat*), longitude (*lon*), and altitude (*alt*), while virtual locations are characterized by the *Edge* and *Cloud* type. When a node is placed in the *Cloud* or on the *Edge*, its connectivity QoS are inherited from the slice *Backhaul* and *Midhaul*, respectively. Except from the physical nodes, virtual nodes can also have a geographical location only if it is the same as a RU location that the aforementioned service will be deployed with. For instance, Figure 6 illustrates a server that will be deployed along a RU with coordinates $(52.5292461, 13.3786919, 15)$.

D. Mobile Node Geo-Positioning and Trajectory Updating

Up to this point, the modeling introduces all necessary abstractions for the emulation of the execution environment and networking, but considers a fixed positioning for all entities. However, in today's 5G era, network entities (i.e., connected vehicles) are not fixed and rather move across the area of operation. To overcome this limitation, 5G-Slicer enables users to annotate the description of a *Node* with position updates that will be realised during emulation. Towards this, users can annotate the description of a (mobile) Node with a *trajectory*. A trajectory is a timestamped list of geo-locations across the emulated spatial plane. Hence, the model considers that the continuous movement of a mobile node can be split into discrete timestamped location updates.

Upon execution, 5G-Slicer will apply for each step in the trajectory a *Move* action, that takes as input the mobile node and the new destination. 5G-Slicer instantly transfers the given node and updates the QoS of the node's network connectivity. Figure 7 depicts a hypothetical trajectory for *UE-bus-1* of the running example, where 5s in the scenario, 5G-Slicer will "move" the bus to the geo-location with coordinates $(52.51263, 13.38948, 0)$. At this point, the node connectivity is updated by invoking the *distance-to-QoS process* and, hence, the bus data transfer rate may be (significantly) altered. Similarly, 5s later, a new move action is invoked and the bus changes location, again. Nonetheless, annotating thousands of nodes with numerous positioning updates requires significant manual effort. 5G-Slicer also provides users with the ability to annotate a node with a *trajectory model* that will dynamically derive the geo-positioning of a node at runtime based on collected monitoring data.

V. Implementation Details

A. The Network Conceptual Graph (NCG)

The NCG is an in-memory undirected graph capturing a valid snapshot of the topology during execution. Upon deployment, the *Topology Builder* introduces every physical and virtual component, as a node in the graph. Each node is annotated with a name, type (network or compute), and optional key-value properties. RUs take the network node type if they lack the ability to perform compute tasks. In contrast, compute nodes (e.g., UE, VMs hosted on MECs), have an identifier for the model blueprint, and node properties can include information provided by the user during the modeling, such as CPU power, core count, memory size, running service name, etc. Network functions, such as packet-level inspection, firewall rules, and traffic policies, can also be attached to a node. The graph edges denote network connections between nodes. For stationary nodes, such as RUs or edge servers, the model provides permanent connectivity, via midhaul or backhaul. These nodes, construct a fully connected Network Core. For mobile nodes, the system produces a list of the RUs when the node falls within their coverage, sorted by the (network) distance to reach each RU. The RU coverage is equal to its radius, as defined in the model description or generated based on the physical characteristics of a network protocol.

The *Topology Builder* populates the generated network QoS parameters originating from the physical layer along with the VNFs to the created edges. Then, the system triggers the mapping and submission process of the topology to the underlying emulation framework primitives. During the experimentation, changing the position of a mobile node does not require the reconstruction of the entire graph but only the connectivity properties of the moving node. Every change to the graph enforces the update of the running emulated infrastructure. Furthermore, the system periodically saves snapshots of the graph. If there is an error (or the user would like to restore the state of a topology), the system will load a stored snapshot

and roll back the whole infrastructure to the saved state. The *NCG* is not only a high-performance structure for keeping topological information, but can also be enriched with real-time metrics from the underlying emulation. This supports a wide range of functions for investigating the structure and dynamics of both networks and applications.

B. QoS for the Graph Edges

The graph edges annotation with QoS updates starts with the midhaul and backhaul links. The system introduces backhaul QoS on the edges between the Cloud nodes and the rest of the network, and midhaul QoS on the edges among the RUs and MECs. The QoS values are provided during the modeling, and as previously mentioned, are considered relatively stable. Next, the system annotates the edges between the RUs and the UE. As previously described, the system provides two model types for UE-to-RU connectivity, namely, the math-based degradation models and physical wireless protocols (i.e., MIMO). The QoS annotation based on degradation models is straight-forward meaning that the system invokes the selected model, passes the UE-to-RU distance as an independent variable, and the model returns the respective QoS.

For wireless channels, we have implemented a proof-of-concept methodology for multi-user MIMO and beamforming systems. Initially, we investigate how distance affects the signal quality and utilize Propagation Loss models to compute receiver signal power by considering the signal power of the transmitter and the antennas' receiver and transmitter positions. Since the formulas of the propagation models are widely used in network simulations, we embrace the propagation models from NS3 [18], which features more than 15 different models, such as Free Space Loss (Friis), Log Distance Propagation, and Three Log Distance Propagation. We set Friis as the default propagation loss model of the implementation, considering it as the most generic and representative model for IoT scenarios (i.e., smart-city, UAVs). With this, the system computes the receiver power (P_r) of a mobile node as follows:

$$P_r = P_t + G_t + G_r - L \tag{1}$$

where P_t is the transmitter output power, G_t is the gain of the transmitter's antenna, G_r is the gain of the receiver's antenna and L represents the path loss. Every parameter is measured in dB. Next, we calculate the signal-to-noise ratio (SNR):

$$\mathrm{SNR}[dB] = P_r - N \tag{2}$$

where N is the noise in dB, which is calculated via the power spectral density theorem:

$$N = N_0 \times B \text{ , } where \ N_0 = k \times T \tag{3}$$

N_0 is the thermal noise, calculated as the multiplication of Boltzmann's constant ($k = 1.38 \times 10^{-23}$), and the temperature of the receiver system (T) in Kelvin. So the noise N is equal to N_0 multiplied by the bandwidth (B) of the respective channel. For typical temperatures, the noise in dB can be calculated with the following equation:

$$N[dB] = -174 + 10\log_{10}(B) \tag{4}$$

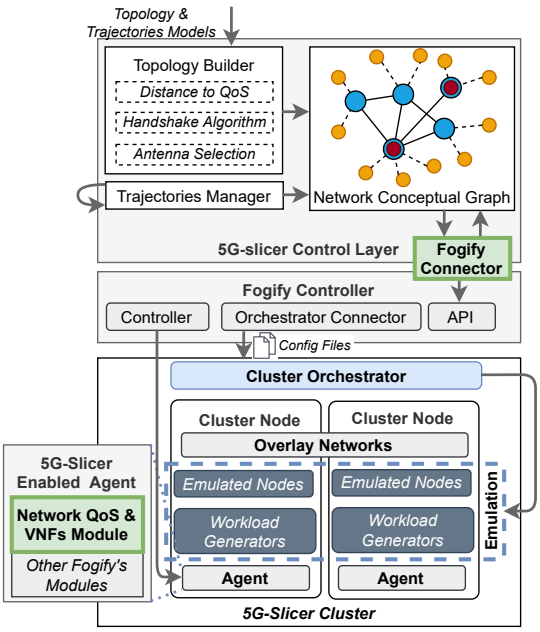

Fig. 8: 5G-Slicer and Fogify Integration Overview

Having the computed SNR, the system needs to calculate the respective high-level network QoS. We calculate the data rate by utilizing the Shannon-Hartley theorem for each device; thus, the spectral efficiency is:

$$C_i = \min(N_t, N_r) \times B \times \log_2(1 + \mathrm{SNR_i}) \tag{5}$$

where C_i is the expected capacity of the channel between the i-th device and the base-station, given the channel bandwidth B in Hz, N_t the number of transmitter's antennas, and N_r the number of receiver's antennas. Taking into account a typical multiuser MIMO (Figure 3), where the RU is implemented with N antennas, and serves M concurrent devices with every device to have k_i antennas, we conclude to the following:

$$\min(N_t, N_r) = k_i, \text{ if } N \geq \sum_{i=1}^{M} k_i \tag{6}$$

If the devices exceed the capacity of N station's antennas, a new device can be connected to a near-by base station, if it is possible; otherwise, the devices will not be connected.

Additionally, for the packet error rate (PER), we utilize the NS3 models. NS3 provides a wide range of error rate models tailored for specific protocols, such as the direct-sequence spread spectrum (DSSS) model. The error models of the system take as input only the protocol specification spectrum and the calculated SNR, and return the packet error rate as a percentage. Moreover, we apply an additional small network delay (e.g. $1ms$) to provide a fronthaul network effect (links among RAN components). Finally, 5G-Slicer proceeds with the annotation of the respective RU-to-UE graph edges.

C. Integration with Fogify Emulation Framework

With the annotation process completed, the Emulator Connector translates the conceptual graph into an emulated execution runtime. For the current version of 5G-Slicer, the

Emulator Connector interface is implemented for Fogify, a scalable emulator with modeling abstractions for the underlying fog offerings of IoT microservices [10]. We selected Fogify because it can facilitate the scalable emulation of distributed computing with support for software-defined networking and run-time ad-hoc alterations to both the emulated instances and the experiment configuration. Figure 8 depicts the integration with Fogify. Initially, the Connector translates the conceptual graph into the modeling primitives of Fogify and submits them to Fogify's API. To do so, we created a set of functions that programmatically build the Fogify model, perform the submission of it, apply ad-hoc emulation changes, and manipulate the whole execution. When Fogify receives the model description, if no error is discovered, translates the model specification to the underlying orchestration primitives. Then, it deploys them via the Cluster Orchestrator, ensuring the instantiation of the containerized services on the emulated environment.

Located on every cluster node, Fogify Agents expose an API to accept requests from the Fogify Controller, apply network QoS primitives, and monitor the emulated devices. We overrode Fogify's network QoS enforcement mechanisms to be 5G-Slicer-enabled, encapsulating fine-grained link QoS variability, VNFs, and packet-level monitoring (SectionV-D). What is more, on a running deployment, Fogify permits users and programs to perform *Actions* and "what-if" *Scenarios* (sequences of timestamped actions) on their IoT services, such as ad-hoc faults and topology changes. When an Action or a Scenario is submitted, Fogify coordinates its execution with the Cluster Orchestrator and the respective agents. 5G-Slicer takes advantage of this functionality to enforce connectivity changes that arise due to the movement of emulated UE entities. When a mobile entity changes its position, 5G-Slicer calculates the alteration of its connection and enforces the respective action to the emulated topology. Finally, 5G-Slicer utilizes the Fogify monitoring system to retrieve performance and app-level metrics from running emulated deployments.

D. Enforcement of the QoS & VNFs

A more detailed view of the *network QoS & VNFs Module* of Figure 8 is provided in Figure 9. To realize the interconnection between emulated nodes, we rely on Fogify's approach that utilizes the VXLAN [19] protocol. Specifically, every network slice is realized as a software-defined overlay mesh network. On every physical host, the system creates a new network namespace, which is composed of an isolated virtual network stack and a virtual bridge. On the virtual bridge the VXLAN tunnel endpoint is connected along with every local emulated node. The nodes communicate with each other locally, through the bridge, or with nodes located on other hosts via VXLAN tunneling. To connect an emulated node to the bridge, the emulator instantiates a virtualized Ethernet adapter (veth) in the VXLAN's namespace and connects it to the virtual bridge. The veth is mapped to an ethernet endpoint inside the container. Then, we introduce 5G-Slicer's mechanisms for firewall rules, packet inspection & monitoring, and

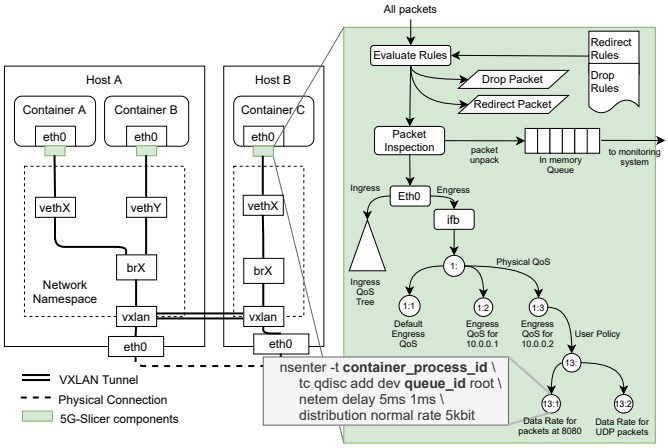

Fig. 9: 5G-Slicer-enabled network QoS and VNF module

fine-grained network QoS enforcement to offer the required functionalities of VNFs, as virtualized UPFs of a slice.

Specifically, when a packet reaches the ethernet endpoint of an emulated node, the 5G-Slicer enabled agent applies the firewall rules provided by the user at the modeling phase. The system has already disseminated the rules to the agents, and the agents utilize the Linux kernel iptables when a new emulated node is instantiated. The Linux kernel processes incoming packets and evaluates, sequentially, every rule in the iptables ruleset (a set of respective firewall rules). If the packet matches the criteria of a rule the Linux kernel decides whether to `ACCEPT`, `REDIRECT` or `DROP` the packet.

If a packet passes the firewall rules, then it travels through the packet inspection module. Since packet inspection is resource-consuming, we allow users to disable this functionality on selected nodes. For packet inspection, the agent starts a new packet processing thread for the respective emulated node. This thread captures every packet and unpacks it to a vector *<source IP, source port, destination IP, destination port, protocol, size, timestamp>*. Then it publishes the vector into an in-memory thread-safe queue. To minimize both the performance overhead and the size of the stored data, the agent periodically exports vectors, summarizes them into time-range groups, and stores them into the Fogify data store.

After packet inspection, the system continues with traffic shaping. The agent instantiates an intermediate functional block device (ibf interface), which is employed to separate and redirect the egress and ingress traffic. Next, it enforces the network QoS by building a tree-based structure of packet filters that steers the packet flow to the respective network QoS queue. To build the tree-like structure, through classful queuing disciplines (qdisc)[2]. The tree root is a network interface, virtual or physical, that the traffic reaches. Nodes are class filters and leaves represent different QoS queues. Every tree entity has a unique identifier, which helps the system to select and apply filtering and QoS to specific nodes and leaves, respectively. The traffic traverses the tree nodes by following the packet filtering, and, in the end, reaches a leaf of the tree. Every leaf assigns the arrived packages to a network queue

[2] http://linux.die.net/man/8/tc

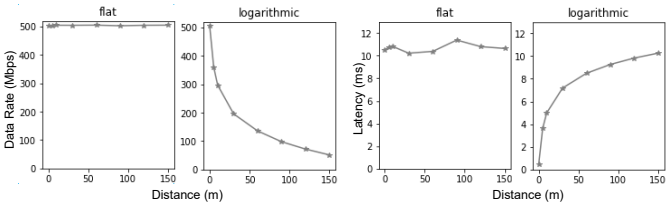

Fig. 10: Flat and logarithmic Network QoS

Parameter	value
Carrier frequency (GHz)	28
RU signal bandwidth (MHz)	100
Transmit power (dBm)	30
UE noise figure (dB)	7.8
RU MIMO antenna array config	8x8
UE MIMO antenna array config.	4x4
MIMO RU antenna element gain (dBi)	6
MIMO UE antenna element gain (dBi)	6
Min SISO bitrate (Mbit/s)	53.87
Max SISO bitrate (Mbit/s)	538.71

TABLE I: Parameters of the Experiment

with the desired QoS characteristics by utilizing the Linux NetEm[3]. At this point, the filtering provided by the system is performed only on the level of a packet IPs without taking into account more fine-grained filter policies, such as QoS policies for specific protocols, ports, etc. To overcome this limitation, we extended the tree-based structure with one extra level of user function plane policies. Thus, if the user has selected to apply an extra policy, the 5G-slicer enabled agent creates an extra tree leaf node where it redirects packets that are compliant with the respective policy.

VI. Evaluation

This section provides a comprehensive evaluation of the 5G-Slicer feature set. Initially, experiments are performed to evaluate the realism of 5G-Slicer to apply network QoS by using mathematical models and the MIMO wireless protocol for signal degradation. Furthermore, we evaluate the applicability of VNFs and packet-level monitoring. Next, we introduce a city-scale scenario of an application utilizing real-world data. The scenario is realized through 5G-Slicer programming primitives and is exported as a reusable template. Finally, based on this scenario, we evaluate the scalability of 5G-Slicer in terms of network and compute resources.

A. Radio Connection Models & VNF Enforcement

1) Radio Connection Models: Initially, we investigate how 5G-Slicer applies different connectivity models between the RU and UE. We utilize iperf3[4] and Linux ping[5] to measure the network data rate and latency. For thoroughness, we present the mean of 5 bandwidth trials, and ping measurements for 500 intervals. Starting with the mathematical models, we examine the results of distinct functions, namely flat and logarithmic. We set the radius of the RU signal coverage to 150m. Figure 10 depicts the data rate and latency for the functions with different distances between the RU and a UE. One clearly observes that the measurements follow the enforced mathematical functions. With a more detailed view of the results, we can see that the

[3] http://linux.org/docs/man8/tc-netem.html [4] http://iperf.fr/iperf-doc.php
[5] http://linux.die.net/man/8/ping

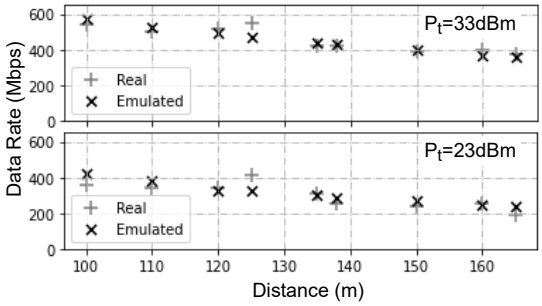

Fig. 11: Comparison of Real & Emulated connectivity

values for the emulated data rate presents a deviation of at most 5%. This is in line with the findings of the Fogify paper [10] where the data rate experienced a 3-5% overhead due to the additional virtualization layer in the network stack.

2) Emulation Realism: To evaluate 5G-Slicer emulation accuracy, we used data-rate measurements provided by the operators of a 5G testbed installation [20], [21]. The testbed captures data rate measurements for gNB-to-UE connectivity in a mixed landscape for various distances (100 to 165m). The testbed features a LimeNET 5G-SA BTS and Huawei P40 Pro that are used as the gNB and UE, respectively. The connectivity is achieved via a 100MHz band channel operating at 3489.42 MHz with a MIMO 2x2 configuration. The RU was set to transmit with two transition power configurations (P_t), namely 23dBm and 33dBm. The antenna gain for the gNB is 15.3dBi and the noise figure is 7dB when the RU operates at 23dBm and 16dB when the RU functions at 33dBm (manufacturer reference values). Figure 11 illustrates the measured data rate for the real and emulated deployments. From this, we observe that the 5G-Slicer results follow the distribution of the real testbed with the mean absolute percentage error being 11.7% for 23dBm and 5.3% for the 33dBm configuration. These modest error rates are observed even with the presence of outliers in the real measurements. Specifically, there is a significant deviation between the real and emulated measurements at 125m for both configurations. This is attributed to the positioning of the UE, where for this particular distance a physical obstacle is influencing the quality of the data rate. This is reasonable because the theoretical models do not consider obstacles, ambient noise, signal inflections, etc., that can influence signal quality.

Next, we evaluate the performance of embracing the MIMO wireless protocol in different configurations. Since these models have a static network and modest network delay, we focus only on the bandwidth. Table I depicts the maximum and minimum bit-rate, with these values fully compliant with the 3GPP guidelines for network slicing [17], [22]. For the one-way network latency (including system delay), we set the minimum and maximum values to $0.1ms$ and $5ms$, respectively. We start the evaluation from a simple 1x1 SISO (single-input-single-output) channel, and we continue with 8x2, 8x4, 8x6, and 8x8 antenna configurations. In Figure 12 (left), we see the performance for the antenna configurations for different RU-to-UE distances. The results are in respect to equations (5) and (6), where, for instance, the capacity of the 8x2 configuration

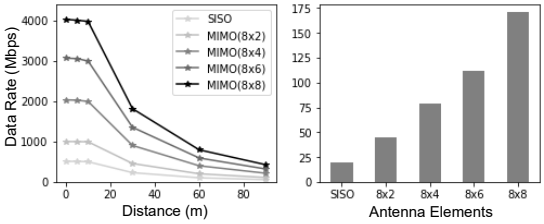

Fig. 12: Measure Bandwidth for SISO & MIMO channels

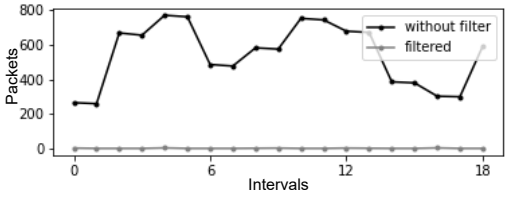

Fig. 13: Traffic With and Without Packet Filtering

is twice as large as the SISO channel. Figure 12 (right) depicts the average error between the measured and computed values. The error follows the bandwidth size, with the SISO channel having the least difference between enforced and observed bandwidth, contrary to the 8x8 configuration, with the maximum bandwidth fluctuation. Still, the percentage difference is less than 6% in every case. To this end, *5G-Slicer correctly computes the radio QoS by following the mathematical or theoretical models, but the underlying emulation framework plays a significant role in their precise enforcement.*

3) VNF Application: For this experiment, we deploy an object classification service (YOLOv3[6]) on an emulated instance with 4 cores@2GHz and 4GB RAM. A simple workload generator (client) is created to disseminate sample images every 5 seconds to the deployed service through a 5G slice. We run the experiment twice for 6 minutes, once with the default configurations (Table I), and once with network-level packet filtering enabled. For the latter, we filter all packets reaching the port 8080. In both tests, we enable the packet level monitoring and set the periodicity to be 15 seconds. Hence, we evaluate both the packet level monitoring and enforcement of VNFs. Figure 13 depicts the number of packets that are transmitted from the client to the classification service. We see that when the filtering is applied, the transmitted packets are almost zero. There are only a few packets streamed from the client for the initial HTTP request handshake process. However, since the packets do not reach the service, the service does not respond. In detail, 3.8M packets reached the server and 200k packets reached the client without filtering. On the contrary, only 660 packets reached the server and 0 the client when the filtering is enabled. In conclusion, *5G-Slicer is capable of applying both the described VNF and packet-level monitoring on an emulated 5G slice*.

B. ITS Operator Use-Case

This section showcases the usability of 5G-Slicer along with a representative benchmark analysis that reveals hidden insights for a mobility-aware IoT application. Let us consider

[6] https://pjreddie.com/darknet/yolo/

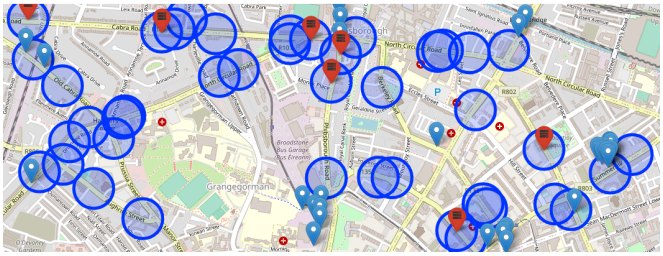

Fig. 14: 5G-Slicer Interactive Map

a scenario with a bus company that intends to analyze real-time data from its fleet to provide better services for on-time bus arrivals (Figure 3). For this, the company does not deploy its own infrastructure but rather leases a 5G network slice from a Network Operator. The company purchases GPS tracking devices and creates a service to periodically send data from buses to an Intelligent Transportation Service (ITS). The ITS consists of two components, namely, an *edge service* that captures and generates neighbor-range analytics, and a *cloud service* that produces overall analytic results. As an exemplary analytic query, consider the mean delay of all buses for the last 5 minutes. The entities of the experiment are the following:

- **IoT device**: reproduces the data generated by each bus, simulating the IoT data dispatch to the nearest MEC. If the *IoT device* is not in the range of any RU, it stores the data until re-connected to the network. Every generator produces a static rate of 10 req/sec. *IoT device* is emulated with 1 core@500MHz and 512MB RAM;
- **MEC Server**: stores the incoming data, computes area-based analytics, and forwards the results to the centralized server. It provides 4 cores@1.4GHz and 4GB RAM.
- **Centralized Server**: is placed in the Network Core, gathers partial data, and calculates the final results. The centralized server has 4 cores@2.4GHz and 4GB RAM.
- **Radio Units**: we consider the parameters from Table I to emulate the radio access connectivity.

Regarding the implementation of the use-case, we utilize the programming primitives provided by 5G-Slicer, and we realize the use-case as an extension of the use-case templates interface. Specifically, we use the real-world open-access dataset from Dublin's bus traces [7], and bus stops[8] as IoT device traces, and possible locations for 5G MIMO antennas, respectively. The datasets include more than 950 bus traces and over 4000 bus stops, with each datapoint of the bus trace featuring 16 metrics, including bus id, location coordinates, operating city region, etc. At the initialization of the use-case, the user defines parameters like the bounding box of interest, and the number of RUs, traces, and MEC servers. The system randomly selects the aforementioned elements from the datasets, and deploys the application automatically. Figure 14 depicts an example of the 5G-Slicer use-case map from a $1.139km \times 3.976km$ bounding box with 50 RUs and their radius (blue circles), 7 MEC servers (red points), and 30 buses (blue points). For the sake of brevity, we use this bounding box in all experiments.

[7] https://data.smartdublin.ie/dataset/dublin_bus_sample
[8] https://data.gov.ie/dataset/bus-stops-served-by-dublin-bus

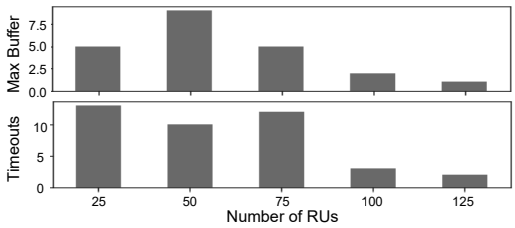

Fig. 15: IoT Device Buffer Size and Timeout Count

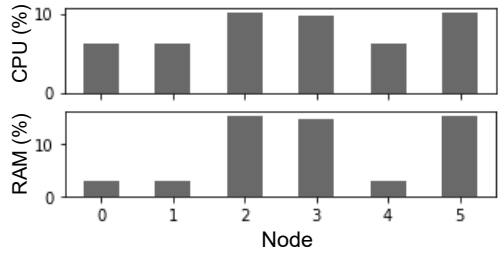

Fig. 16: CPU and Memory utilization

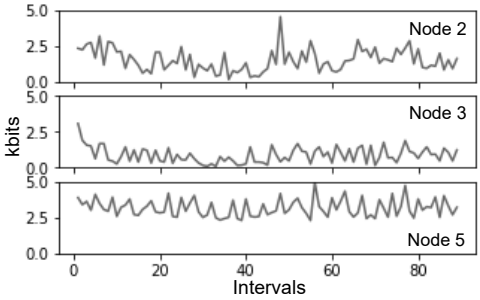

Fig. 17: Network utilization

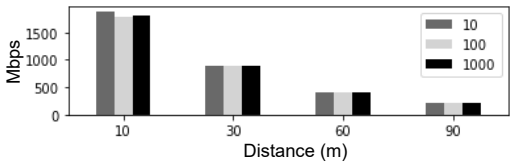

Fig. 18: Network Access Elements Scalability

First, we evaluate how the cardinality of RUs influences the needs of IoT devices for memory and data freshness. We select a random bus ($id = 1230$) moving through the route with id 33568, and we alter for each test the number of RUs (25, 50, 75, 100, 125). Figure 15 depicts the maximum queued datapoints and the number of timeouts that the IoT device has experienced during the execution. From 25-75 RUs, the maximum buffer, which an IoT device needs, ranges from 5 to 8 elements while for 100 and 125 the value is lower than 2. Furthermore, the count of the timeouts (when a timeout occurs the IoT queued the datapoint to its buffer) is more than 10 for less than 100 RUs contrary to less than 5 when the number of RUs is more than 100. The latter indicates that *the density of the RUs should be at least 100 in the selected bounding box to have fresh and in-time results*.

The next test investigates the performance of the deployed application. We opt to emulate one cloud server, 6 MEC servers deployed on randomly selected base-stations, and 20 IoT devices in the same city area. In this experiment, the IoT devices move through the city with 5G-Slicer enforcing the trajectory updates. After the experimentation, we retrieve the monitoring data from the MEC nodes to investigate performance and placement issues. Figure 17 illustrates the CPU and Memory utilization of the deployed services. We notice that nodes with ids 2, 3, and 5 have 10% CPU utilization on average compared with nodes 0, 1, and 4, which have around 5% CPU utilization. The CPU difference is followed by a similar memory utilization difference, with the first group of nodes (2,3,5) having around 15% memory utilization contrary to the others (0,1,4) that occupied only 3% of the memory. To have a clear overview of the issue, we investigate network slice traffic. The investigation reveals that there is no IoT device in the reach of the 0, 1, and 4 MECs during the experimentation. Furthermore, Figure 17 depicts the network traffic in bytes for the rest of the edge nodes. Interestingly, even if the nodes have similar utilization for CPU and memory, the network traffic is fluctuating between the nodes. Specifically, node 5 exchanged most of the data during the experiment, while nodes 2 and 3 transferred much fewer bytes. Our conclusion is three-fold: *(i) the placement of the Edge service should be in line with the IoT devices density, (ii) generally, the MECs were under-utilized during the experiment, and (iii) MECs have stable CPU and memory utilization independently from the incoming traffic*.

C. Scalability Evaluation

In this section, we test the scalability of 5G-Slicer. Since the network access elements of the system are realized only on the conceptual graph, we initially examine if this scheme is scalable. Then, we perform a scalability analysis for the system's limits regarding the underlying infrastructure.

1) Network Access Elements Scalability: In this experiment, we deploy a MEC and the IoT generator of the previous test in different city areas. In the same bounding box, we deploy 10, 100, and 1000 RUs, and we perform a similar bandwidth experiment as described in the Section VI-A1 for MIMO with an 8x4 antenna configuration, but with the distances between the MEC and IoTs to be 10, 30, 60, and 90m. Since the midhaul bandwidth is much higher than the bandwidth of the radio access connections, we expect radio bandwidth to dictate the bandwidth between the MEC and IoTs. Interestingly, Figure 18 shows that the values for the different distances follow the same distribution as the Section VI-A1 experiments. The latter is reasonable because only the conceptual graph keeps the information about the RUs without any compute resource provisioning with the memory required for 1000 nodes amounting to 1.3GB. Thus, the only "limiting" condition of the system is the size of the conceptual graph. Running on a regular PC, 5G-Slicer supports thousands of RUs without any issue. So, *the system has no performance degradation originating from the number of RUs*.

2) Scalability of Compute Components: In this experiment, we deploy 5G-Slicer on top of three different clusters, namely, a single-node cluster, 2-node cluster, and 3-node cluster. Each node is a VM equipped with 16cores@2.4GHz, and 16GB of memory. Based on the description of the emulated instances capabilities of the use-case in Section VI-B, we opt to have

Fig. 19: Bootstrapping Duration and Scalability

1/2/8 emulated nodes for cloud server, MEC servers, and IoT workload, respectively, on a single cluster node. The latter deployment reaches the limit for an accurate emulation based on memory and the 65% of the host's available processing power. Consequently, the framework should be capable of emulating 2/4/16 and 3/6/24 nodes on the 2-node and 3-node clusters. Indeed, the framework is able to instantiate the aforementioned instances. Next, we opt to evaluate the generality of our solution, we reduce the capabilities of IoT by half. 5G-Slicer achieves to double the number of the IoT devices with only overhead at the emulation's bootstrapping time. Figure 19 highlights the bootstrapping duration in seconds for the different deployment schemes (*Small* or *Large Deployment*). The deployment with 57 emulated nodes (3/6/48) had the higher deployment time with 290secs, contrary to 11 (1/2/8) nodes emulation, which took 67secs for bootstrapping. It is obvious that the bootstrapping duration is proportional to the number of the emulated nodes. In conclusion, *the 5G-Slicer is only bounded by the resources reserved for the underlying execution environment, and when the resource pool expands, the emulation capabilities proportionally increase with a comparable raising of the bootstrapping time.*

VII. Related Work

To date, the majority of the efforts towards rapid experimentation of 5G networks introduce simulators adopting theoretical models for the behavior of various network infrastructure components. For instance, Muller et al. [8] introduce the Vienna 5G simulator that enables the modeling of 5G networks with arbitrary geometry and several types of base stations with various performance models for evaluating signal strength and interference. In turn, Mezzavilla et al. [9] introduce a 5G simulator that targets the evaluation of services and control algorithms that embrace the mmWave band. Moreover, Nardini et al. [23] introduce Simu5G, a simulator targeting the performance evaluation of the data plane of 5G services deployed over the radio network by providing abstractions for various user equipment and gNBs. Nonetheless, the aforementioned approaches only take into account the modeling of various components of the 5G infrastructure layer with limited or no provisions for modeling node mobility, MECs, VNFs, or SDNs in the form of network slices.

A key limitation of simulators is that behavior models often ignore unforeseen system-level effects of the application, especially in complex realms with multiple interacting entities. Emulators attempt to tackle this pressing challenge by enabling the actual application deployment, yet, mimicking the production environment at a significantly lower cost [10]. Hardware-based emulators, like Colosseum [24] or AERPAW [25], provide realistic emulation, scalable networking, and processing capabilities. However, their "hardware-in-the-loop" approach increases the cost, the implementation effort and requires remote access to external infrastructures. Moreover, various edge computing software-based emulators originate from the extension of network emulators, such as MiniNet [26], in an attempt to introduce experiment testbeds with compute node and network link heterogeneity [27]–[29]. Focusing on scalability and ad-hoc runtime alterations, Fogify [10] and MockFog [30] take advantage of distributed cloud resources to instantiate large-scale multi-host experiments. Interestingly, MeDICINE [11] is a VNF prototyping platform that is able to execute network functions, provided as software containers, in an emulated compute environment. Nevertheless, just as with the simulators, little or no provision is made in introducing a multi-host emulator for 5G-enabled networks encapsulating key components of the virtualization layer, such as network slicing, VNF enforcement and positioning of mobile entities.

VIII. Conclusion

In this work, we introduce 5G-Slicer, an open and extensible framework for the design and rapid experimentation of IoT applications on top of emulated 5G network slices. 5G-Slicer encapsulates powerful modeling abstractions, and ready-to-use templates, for the description of 5G network infrastructure and key virtualization technologies including VNFs, cloud and edge resources, and the geo-positioning of mobile entities. Through reproducible experiment scenarios, or even at runtime, users can apply configuration changes, inject workload, faults, alter the trajectory of mobile nodes and monitor the resource utilization and network QoS of all emulated components. With a comprehensive evaluation, we show that for even complex testbeds, 5G-Slicer presents at most an 11.7% error when comparing actual and emulated network QoS. Last but not least, the scalability of 5G-Slicer is independent of the number of radio units and is only bounded by the emulated compute nodes. Still, as 5G-Slicer can be deployed in a multi-host environment, when the underlying resource pool increases, so does the scalability of 5G-Slicer.

Our future work includes the evaluation of emulation performance in terms of connectivity update delay along with a more extensive scalability testing, creation of emulation connectors with other network simulators and emulators, such as NS3, or mininet, and the design and implementation of edge native services and concepts such as O-RAN intelligent controllers, mobility northbound APIs, etc.

Acknowledgement. This work is partially supported by the EU Commission through RAINBOW 871403 (ICT-15-2019-2020) project and by the Cyprus Research and Innovation Foundation through COMPLEMENTARY/0916/0916/0171 and INFRASTRUCTURES/1216/0017 (IRIDA) projects. The authors wish to thank Dr. Christos Tranoris and prof. Spyros Denazis of the U. of Patras for providing measurements from the "Patras 5G" testbed, which was supported by the 5GVINNI H2020 (EU grant agreement No. 815279)

REFERENCES

[1] H. Zhao, S. Deng, Z. Liu, Z. Xiang, J. Yin, S. Dustdar, and A. Zomaya, "Dpos: Decentralized, privacy-preserving, and low-complexity online slicing for multi-tenant networks," *IEEE Transactions on Mobile Computing*, 2021.

[2] V. Sciancalepore, F. Cirillo, and X. Costa-Perez, "Slice as a service (SlaaS) optimal IoT Slice resources orchestration," *GLOBECOM*, 2017.

[3] D. Trihinas, G. Pallis, and M. Dikaiakos, "Low-cost adaptive monitoring techniques for the internet of things," *IEEE Transactions on Services Computing*, 2021.

[4] X. Shen, "Device-to-device communication in 5g cellular networks," *IEEE Network*, vol. 29, no. 2, pp. 2–3, 2015.

[5] J. Ni, X. Lin, and X. S. Shen, "Efficient and secure service-oriented authentication supporting network slicing for 5g-enabled iot," *IEEE Journal on Selected Areas in Communications*, 2018.

[6] P. Vitello, A. Capponi, C. Fiandrino, G. Cantelmo, and D. Kliazovich, "Mobility-Driven and Energy-Efficient Deployment of Edge Data Centers in Urban Environments," *IEEE TSUSC*, 2021.

[7] M. Richart, J. Baliosian, J. Serrat, and J. L. Gorricho, "Resource Slicing in Virtual Wireless Networks: A Survey," *IEEE TNSM*, 2016.

[8] M. K. Müller, F. Ademaj, T. Dittrich, A. Fastenbauer, B. R. Elbal, A. Nabavi, L. Nagel, S. Schwarz, and M. Rupp, "Flexible multi-node simulation of cellular mobile communications: the Vienna 5G System Level Simulator," *EURASIP Journal on Wireless Communications and Networking*, Sep. 2018.

[9] M. Mezzavilla, M. Zhang, M. Polese, R. Ford, S. Dutta, S. Rangan, and M. Zorzi, "End-to-end simulation of 5g mmwave networks," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2237–2263, 2018.

[10] M. Symeonides, Z. Georgiou, D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Fogify: A fog computing emulation framework," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, 2020.

[11] M. Peuster, H. Karl, and S. van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016.

[12] S. Wang, K. Chan, R. Urgaonkar, T. He, and K. K. Leung, "Emulation-based study of dynamic service placement in mobile micro-clouds," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, 2015.

[13] R. A. Addad, M. Bagaa, T. Taleb, D. L. C. Dutra, and H. Flinck, "Optimization model for cross-domain network slices in 5g networks," *IEEE Transactions on Mobile Computing*, vol. 19, pp. 1156–1169, 2020.

[14] S. A. Busari, K. M. S. Huq, S. Mumtaz, J. Rodriguez, Y. Fang, D. C. Sicker, S. Al-Rubaye, and A. Tsourdos, "Generalized hybrid beamforming for vehicular connectivity using thz massive mimo," *IEEE Transactions on Vehicular Technology*, 2019.

[15] "Published o-ran spec." https://www.o-ran.org/specifications, 2020.

[16] P.-H. Kuo and A. Mourad, "Millimeter wave for 5g mobile fronthaul and backhaul," in *2017 European Conference on Networks and Communications (EuCNC)*, 2017.

[17] "Release 16, 3gpp standard," https://www.3gpp.org/release-16.

[18] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34.

[19] M. Mahalingam, D. G. Dutt, K. J. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks," *RFC*, vol. 7348, pp. 1–22, 2014.

[20] C. Tranoris and S. G. Denazis, "Patras 5g: An open source based end-to-end facility for 5g trials," *ERCIM News*, 2019.

[21] D. Giannopoulos, P. Papaioannou, L. Ntzogani, C. Tranoris, and S. Denazis, "A holistic approach for 5g network slice monitoring," in *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, 2021, pp. 240–245.

[22] M. T. Moayyed, F. Restuccia, and S. Basagni, "Comparative performance evaluation of mmwave 5g nr and lte in a campus scenario," in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, 2020.

[23] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis, "Simu5g–an omnet++ library for end-to-end performance evaluation of 5g networks," *IEEE Access*, vol. 8, pp. 181 176–181 191, 2020.

[24] L. Bonati, P. Johari, M. Polese, S. D'Oro, S. Mohanti, M. T. Moayyed, D. Villa, S. Shrivastava, C. Tassie, K. Yoder, A. Bagga, P. Patel, V. Petkov, M. Seltser, F. Restuccia, A. Gosain, K. R. Chowdhury, S. Basagni, and T. Melodia, "Colosseum: Large-scale wireless experimentation through hardware-in-the-loop network emulation," in *IEEE International Symposium on Dynamic Spectrum Access Networks, DySPAN 2021*. IEEE, 2021.

[25] M. L. Sichitiu, I. Guvenc, R. Dutta, V. Marojevic, and B. Floyd, "Aerpaw emulation overview," in *Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, ser. WiNTECH'20, 2020, p. 1–8.

[26] B. Lantz, B. Heller, and N. Mckeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *In ACM SIGCOMM HotNets Workshop*, 2010.

[27] Y. Zeng, M. Chao, and R. Stoleru, "Emuedge: A hybrid emulator for reproducible and realistic edge computing experiments," in *2019 IEEE International Conference on Fog Computing (ICFC)*, 2019.

[28] A. Coutinho, F. Greve, C. Prazeres, and J. Cardoso, "Fogbed: A rapid-prototyping emulation environment for fog computing," in *2018 IEEE International Conference on Communications (ICC)*, May 2018.

[29] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures," *CoRR*, vol. abs/1709.07563, 2017.

[30] J. Hasenburg, M. Grambow, E. Grünewald, S. Huk, and D. Bermbach, "Mockfog: Emulating fog computing infrastructure in the cloud," in *2019 IEEE International Conference on Fog Computing (ICFC)*, 2019.