This paper has appeared at IEEE PIMRC 2020, DOI 10.1109/PIMRC48278.2020.9217177

Using Simu5G as a Realtime Network Emulator to Test MEC Apps in an End-To-End 5G Testbed

Giovanni Nardini, Giovanni Stea, Antonio Virdis
Dipartimento di Ingegneria dell'Informazione
University of Pisa, Pisa, Italy
{name.surname}@unipi.it

Dario Sabella
Intel Deutschland GmbH
dario.sabella@intel.com

Purvi Thakkar
Intel Corporation
purvi.thakkar@intel.com

***Abstract*—Multi-access Edge Computing (MEC) allows users to run applications on demand near their mobile access points. MEC applications will exploit 5G infrastructure, and they will have to be designed by taking into account the characteristics of 5G mobile networks. This work describes how to use a system-level simulator of 5G networks – namely Simu5G, which evolves the popular 4G network simulator SimuLTE – as a real-time 5G network emulator. This allows designers of networked applications – and MEC ones in particular – to use it as a testbed during the deployment. We describe the system setup of Simu5G as an emulator, and its emulation capabilities and scale. Moreover, we present a case study of a MEC testbed using Intel's Open Network Edge Services Software (OpenNESS) toolkit, based on a recent demonstration in 5GAA (5G Automotive Association).**

Keywords—Simulation, Emulation, Multi-access Edge Computing, SimuLTE, Simu5G

I. INTRODUCTION

Fifth-generation (5G) cellular networks will bring significant changes to the wireless networking landscape. In fact, they will enable unprecedented ICT-based services, such as smart cities, autonomous vehicles, augmented reality and Industry 4.0. Most of these services will be composed of both *communication* and *computation*, thanks to the deployment of computing and storage capabilities at the edge of the mobile network. An independent, but complementary innovation is in fact represented by Multi-access Edge Computing (MEC), which will endow the mobile network with cloud-computing capabilities, to allow mobile users to leverage the power of complex algorithms such as those based on artificial intelligence. While MEC is independent of the underlying technology (it can already coexist with the current 4G networks, in fact), it is foreseen that the progressive deployment of 5G will be an enabler for more powerful MEC capabilities.

The MEC infrastructure is expected to host third-party distributed applications, which will open a market segment for ME app developers. These developers have a pressing need for instruments for fast prototyping and credible performance evaluation. In fact, some of the services that they will be developing may have stringent latency constraints, such as autonomous driving or factory automation. For these, changes in the network configuration or deployment may have a drastic impact on their timing properties. When engineering apps, developers need to know in advance what to expect from a 5G network in terms of bandwidth and latency, at the very least. On the other hand, MEC infrastructure owners (often 5G operators themselves) will need to assess the performance of the services they are hosting in a controlled environment, so as to, e.g., evaluate alternative deployments or network functions partitioning. There is therefore a need for instruments that allow one to quickly setup a testbed, where the two sides of MEC apps exchange traffic through a 5G network. Unfortunately, 5G network testbeds are hard to come by, especially for developers.

The authors of this paper recently developed Simu5G, a system-level simulator of 5G New Radio networks based on OMNeT++, which evolves from the well-known SimuLTE simulator of LTE/LTE-A networks. In this paper, we show how to configure Simu5G to run as a network emulator, allowing a user to test the performance of real applications when they communicate via a 5G network. These applications can be, for instance, the two counterparts of a MEC app, one running on a 5G User Equipment (UE) in mobility and the other on a MEC host connected to the 5G infrastructure. This allows application developers to test the performance of their software on a 5G network, under controlled conditions (e.g., as for load, channel quality, mobility, etc.) in a pre-production environment, so as to obtain confidence regarding their performance. We describe the setup of Simu5G as an emulator, which includes some non-trivial configurations of the OS networking functions, and we analyze the performance of the emulation, identifying the limiting factors in a network-emulation scenario - e.g., the maximum number of simultaneous users that can be simulated in a scenario, or the maximum traffic throughput that can be carried. Moreover, we describe the setup and configuration of an end-to-end MEC/5G testbed, where Intel's Open Network Edge Services Software (OpenNESS) is used as a MEC host, running applications that interact with the end-user apps on the UE through an emulated 5G network.

The rest of the paper is organized as follows: Section II describes Simu5G and shows how to configure a system to run it as an emulator. Section III evaluates the emulation capabilities of Simu5G on a standard desktop computer. Section IV presents a comprehensive testbed involving Simu5G and MEC applications running on the Intel OpenNESS toolkit. Finally, Section V

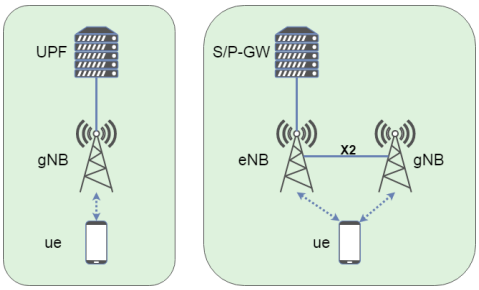

Fig. 1: SA (left) and ENDC (right) deployment

draws conclusions and highlights directions for future work.

II. SIMU5G

A. Description of the simulator

Simu5G [3][4] is the evolution of the well-known SimuLTE 4G network simulator [1][2] towards 5G NewRadio access. It is based on OMNeT++ [5] and it incorporates models from the INET library [6], which allows users to construct end-to-end TCP/IP scenarios, involving routers and end hosts. Simu5G simulates the data plane of both the core and the access networks.

As far as the core network (CN) is concerned, it allows users to instantiate a User Plane Function (UPF) or Packet GateWay (PGW) and an arbitrary topology, where forwarding occurs using the GPRS tunneling protocol (GTP). As far as the radio access is concerned, it allows one to instantiate gNBs and UEs, which interact using a model of the New Radio protocol stack. gNBs can be either connected to the CN directly, as shown in Fig. 1 (left), in the so-called StandAlone (SA) deployment. Alternatively, a gNB can operate in an E-UTRA/NR Dual Connectivity (ENDC) deployment, shown in Fig. 1 (right), where LTE and 5G coexist. This last deployment is expected to be the most common in the early phases of 5G deployment. In this last configuration, the gNB works as a Secondary Node (SN) for an LTE eNB, which acts as Master Node (MN) connected to the CN. The eNB and the gNB are connected through the X2 interface and all NR traffic traverses the eNB first. UEs have a dual stack (LTE and NR), with a Packet Data Convergence Protocol in common to allow in-sequence delivery to the higher layer.

As far as the physical layer is concerned, Simu5G follows the approach already used by SimuLTE, i.e. to model the *effects* of propagation on the wireless channel at the receiver, *without* modelling symbol transmission and constellations. When a sender sends a MAC PDU to a receiver, the two OMNeT++ modules exchange a message. On receipt of said message the receiver performs a series of operations, summarized as follows:

- compute the reception power of the signal on each Resource Block (RB) x occupied by the MAC PDU, starting from the transmission power at the sender and applying a *channel model* to model pathloss, fading and shadowing;
- compute the interference by summing up the power received by all the other senders that interfere on the same RBs (using the same transformation as above);
- compute the SINR on each RB x, using obvious algebra;
- $\forall x$, compute $P_x = BLER(MCS, SINR_x)$, the error probability for that RB given the Modulation and Coding Scheme (MCS) used by the sender and the received SINR. This is done by using Block Error Rate (BLER) curves, obtained from link-level simulators (e.g., [8]);
- compute $P = 1 - \prod_x (1 - P_x)$, the error probability of the whole MAC PDU, extract a sample of a uniform random variable, and test its value against P to check if the reception was correct.

It is shown in [7] that the above modeling reduces the computational complexity of the decoding operation, hence the simulation running time, it improves evolvability, making it easy e.g. to add new modulations, and it still allows arbitrary channel models to be used.

Simu5G simulates radio access on multiple carriers, in both Frequency- and Time-division duplexing (FDD, TDD). Different carrier components can be configured with different FDD numerologies and different TDD slot formats. Moreover, different carrier components can have different channel models. Moreover, it incorporates functionalities already modelled in SimuLTE, e.g. UE handover and network-controlled device-to-device (D2D) communications, both one-to-one and one-to-many. Being based on OMNeT++, it allows one to incorporate models from other OMNeT++ libraries, such as user mobility (e.g., through VEINS [9] or LIMOSIM [10]).

B. Real-time emulation

OMNeT++ is a discrete-event simulation framework, where time advances because *events* are processed: every event carries a firing time, and events are sorted by firing time into an event queue. When the next future event is extracted from the queue, the current simulated time is advanced to that event's firing time. However, OMNeT++ allows one to use (among others) a *real-time* event scheduler, according to which the flow of simulated time is *slowed down* to the pace of real (wall-clock) time. This is only possible if simulated time flows faster than the real time, i.e. if the density of events and their processing time are not such as to overload the system processing capacity. The above condition depends on the hardware/software system, on how a simulator is coded, but also on the particular scenario being run. Typically, there will be a *scale* in terms of number of UEs, gNBs, or traffic transmitted within a 5G network, after which a given simulation will not be able to run in real time.

Moreover, the INET library[1] comes with modules that act as a bridge between the simulation environment and the real network interfaces in the host operating system. Packets received by the real interfaces appear in the simulation, whereas simulated packets sent to the latter are sent out on the real network interface. To do this, the INET library provides a network interface module, called *ExtInterface*, which has to be added to the simulated network devices that need to receive/send packets from/to the host operating system. The *ExtInterface* modules capture packets using the PCAP library [11], which makes a copy of packets entering the real network interfaces and stores them into a buffer. An emulation-enabled real-time scheduler is responsible for fetching such packets from the PCAP buffers, to convert them into the equivalent C++ object representation used

[1] In this paper, we refer to version 3.6.4 of the INET library

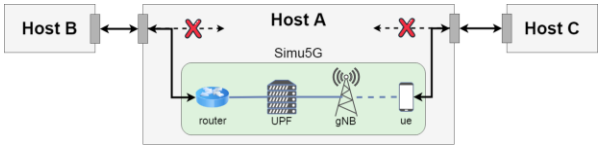

Fig. 2: Simu5G as emulator

in the simulation, and to add them to the event queue, where they are processed like other events generated within the simulation. However, the real-time scheduler fetches new packets from the PCAP buffer *only* when the simulation time is in line (henceforth, *coherent*) with wall-clock time. When the simulation time is slower than the wall-clock time, real packets stay in the PCAP buffer and accumulate delay, until coherence is resumed. Moreover, if real packets arrive faster than the rate at which the scheduler drains the PCAP buffer, the latter fills up and new packets are discarded. When packets need to be sent outside the simulation environment, they are transmitted to the real network using raw sockets. The above features provided by OMNeT++ and the INET library allow one to use Simu5G as a network emulator, that transports packets of real applications, delaying them the way a 5G network would.

C. Configuring a system to run Simu5G as an emulator

We now describe the actions needed to configure a system to make Simu5G capture packets from real network interfaces and run a network emulation.

Without loss of generality, we refer to the simple architecture shown in Fig. 2, where Host A runs Simu5G and is physically connected to two hosts, namely hosts B and C, forming two different IP networks. More complex configurations can also be envisaged like, e.g., having one of the hosts located remotely and reachable using a public IP address. The aim is to configure a testbed where the network traffic between two applications running on, respectively, hosts B and C flows through a Simu5G instance running on Host A.

In order to let packets flow between B and C, we need to configure A's operating system (OS) to enable forwarding of IP packets. Moreover, the appropriate routes have to be added to the hosts' IP routing tables so as to forward packets towards the correct outgoing network interface. In more complex scenarios, also new Network Address Translation (NAT) rules may be needed if it is necessary to exit to the public Internet.

Once the traffic path has been set, we need to have packets that reach Host A traverse Simu5G. Since packets captured by Simu5G are *copied*, rather than redirected, to the emulation, we have to prevent the original packets from following the direct path between the two interfaces, as shown in Fig. 2. This is accomplished by adding packet-discard rules to the OS firewall.

We also need to configure Simu5G so that it captures packets from the real network interfaces and routes them within the emulated network. Considering the simple network in Fig. 2, packets coming from Host B are injected into the *router* module of the running instance of Simu5G, whereas packets coming from Host C are injected into the *ue* module. To do this, *router* and *ue* modules are both equipped with an *ExtInterface* submodule. The latter has two configuration parameters to be specified: *i)* the name of the interface the packets are captured from, and *ii)* which packets need to be captured, i.e. based on their 5-tuple. Further implementation details can be found on the Simu5G website [4].

III. Performance Analysis of Simu5G Emulation

As anticipated in the previous section, real-time emulation is only possible if event processing occurs faster than the real time. This depends on the scenario being simulated: for instance, the more UEs are simulated in the scenario, the more events will be triggered just due to their CQI reporting alone. In this section, we evaluate the performance of Simu5G emulation, with the aim to identify which factors constrains the emulation capabilities, and what a user can expect to be able to run on an off-the-shelf desktop computer. To do so, we setup a system where two hosts run a distributed request-response application, whose packets are forwarded through an intermediate host running the emulated 5G network using Simu5G. In particular, one side of the communication acts as a UE of the 5G network, receiving requests from a remote server and sending back responses.

Ideally, to ascertain if and when the emulation is coherent, we should log the system time T_i and the simulated time t_i whenever the event "beginning of TTI i" is fired. When $T_i - T_0 > t_i - t_0 + \delta$, with δ being the measurement tolerance, the emulation can be impaired. Unfortunately, with TTIs being at or below 1 ms, measuring this is impossible in practice – it would imply that the host machine would be serving system calls to obtain the wall-clock time instead of advancing the emulation. Performing the same test at longer periods (say, every N TTIs, $N \gg 1$) is certainly feasible, but inconclusive, since it does not guarantee coherence at each TTI. For the above reasons, we exploit an *indirect* measurement technique, which is both non-invasive and sufficiently reliable. As discussed in section II.B, a characteristic of OMNeT++, un-documented to the best of our knowledge, is that packets from real host's interfaces are delayed – and eventually discarded – whenever the emulation is not coherent. Therefore, by simply counting transmitted/received IP packets at the interfaces and verifying their RTT we can have an indirect coherence assessment: when the emulation is not coherent, the number of transmitted packets will be strictly larger than the number of received packets and/or the RTT will diverge. We are aware that this only implies that the emulation was coherent *at the time of arrival of packets at the interfaces*, which does not necessarily warrant that it was at any other (unobserved) time; however, arrival times at the interfaces are those when coherence matters the most, which makes this method quite reliable.

A. Experimental configuration

The setting of the testbed is shown in Fig. 3 and is composed of three general-purpose computers, namely Host A, B and C, whose hardware details are as follows:

- Host A is a desktop computer equipped with an Intel Core(TM) i7 CPU at 3.60 GHz, with 16 GB of RAM and a Linux Kubuntu 16.04 OS. It is endowed with two 1Gb/s Ethernet NICs which connect it to Hosts B and C;
- Host B is a desktop computer running a Linux Ubuntu 18.04 OS on an Intel Core™ i7 CPU at 3.60 GHz, with 16 GB of RAM and one 1Gb/s Ethernet NIC;
- Host C is an Apple MacBook Pro with a macOS 10.15.3 (Catalina) OS, equipped with an Intel Core(TM) i5 CPU at

Fig. 3: Testbed setup

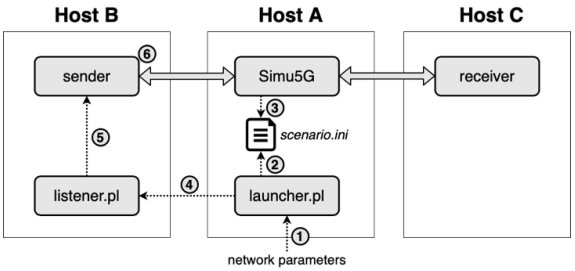

Fig. 4: Launching the testbed

2.40 GHz, 8 GB of RAM and one Thunderbolt-2 NIC endowed with an Ethernet adapter.

The computers are physically connected via Ethernet cables as shown in Fig. 3. In particular, Host B and C are connected to Host A's interfaces called `eth0` and `eth1`, respectively. The Simu5G network emulation is run on Host A, which installs OMNeT++ version 5.3 and the INET library version 3.6.4. The emulated network scenario is depicted in Fig. 6 and includes one gNB and one UE, called *realUe* in the figure. The latter is the counterpart of Host C in the emulation as it is endowed with an *ExtInterface* module to capture packets coming from Host C on `eth1`. Likewise, the router has one *ExtInterface* module that captures packets coming from Host B on `eth0`. With this configuration, packets sent by Host B appear into the emulation at the router and are forwarded towards the UE, which in turn sends them outside the emulation. Then, Host A's OS takes care of forwarding them towards Host C. The reverse path is traversed by packets sent by Host C and directed to Host B. Moreover, a number of *simulated* UEs are added to the network to create a more realistic scenario: these UEs communicate with the simulated server, generating traffic that remains *within* the simulator.

The real network traffic is generated by an application, coded in C++, composed of a sender and a receiver side running on Host B and C, respectively. The two endpoints establish a TCP connection using a socket pair, then the sender generates and transmits periodic request messages. For each request, the receiver replies with a response message. Each request message is tagged with a sequence number and a timestamp so that the Round-Trip Time (RTT) of the communication can be measured upon reception of the associated response. The size L_{req} and L_{resp} of both request and response messages can be configured, as well as the requests' sending interval T.

In order to run the testbed, the receiver application on Host C can be launched at any time, since it will remain idle, listening to incoming connections. On the other hand, the sender application on Host B needs to start the traffic only after the network emulation on Host A has been started. In order to automatize the process, we created a script *launcher* to be run on Host A, coded in Perl, that performs the operations depicted in Fig. 4. It accepts parameters such as the number of simulated UEs, the number of RBs and NR numerology index, and writes them to an INI configuration file. Then, it launches Simu5G, which sets up the network scenario according to the parameters found in the INI file and starts the emulation. Five seconds after the emulation has started, the *launcher* script sends a message via a socket to a *listener* script running on Host B, and the latter triggers the sender application to start the real traffic.

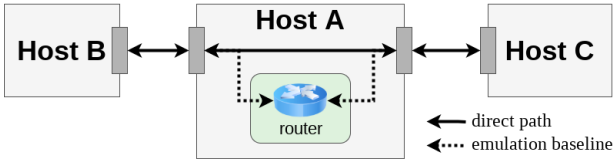

Fig. 5: Baseline configurations

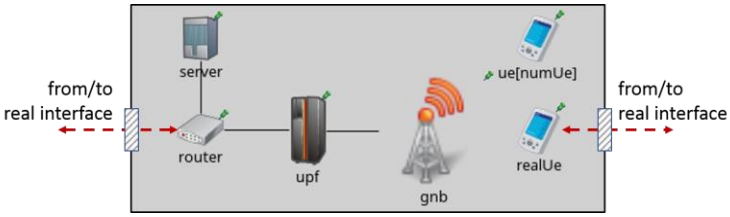

Fig. 6: Emulated network scenario on Host A

TABLE 1 - MAIN NETWORK PARAMETERS

Parameter Name	Value
Carrier frequency	2 GHz
Number of RBs	10
Fading + shadowing	Enabled
gNB Tx Power	46 dBm
gNB antenna gain	8 dBi
gNB noise figure	5 dB
UE antenna gain	0 dBi
UE noise figure	7 dB
CQI reporting period	80 TTIs
Path loss model	[12]
UE mobility	Static
Traffic type	Req-resp

B. *Experiments results*

In the following experiments, we make sure that the number of packets sent and received is the same, and we measure the RTT to understand when the emulation starts struggling, depending on the scenario. We first need to infer a *baseline RTT* to understand when this happens. To do so, we assess the overhead introduced by the OMNeT++ framework to capture packets from the real host's interface and injecting them into the emulation. To do this, we compare the two scenarios depicted in Fig. 5: one (called *direct path*) where Host B communicates with Host C via Host A, which only acts as a router, i.e. it forwards data packets between its two interfaces *without* running any user application. In the other one (called *emulation baseline*) Host A runs a very simple OMNeT++ emulation, consisting of an INET router that forwards packets between its two infinite-speed interfaces. Since the emulated network is minimal, the performance penalty incurred by running the emulation baseline is limited, hence differences between the two scenarios should be accountable to the capturing of packets by OMNeT++. We compare the RTT measured in the two scenarios, by computing the average of the RTT of 300 requests, considering a sending interval $T = 1s$. Since the measured RTT is considerably smaller

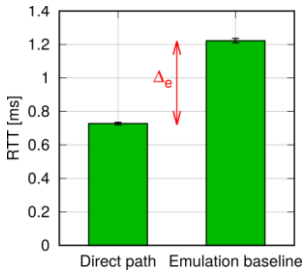

Fig. 7: Emulation overhead

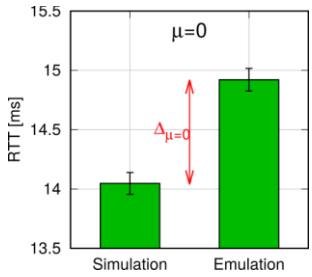

Fig. 8: RTT comparison, $\mu = 0$

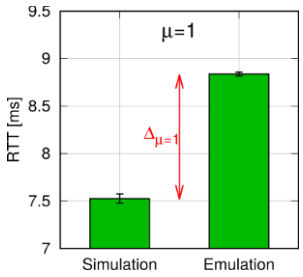

Fig. 9: RTT comparison, $\mu = 1$

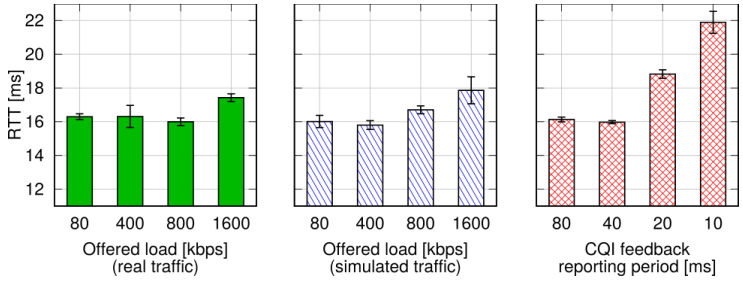

Fig. 10: RTT with varying a) sending interval of real traffic, b) sending interval of simulated traffic, c) CQI reporting period

than T, we can assume that the obtained values are independent and that TCP congestion control plays no significant role. We also set $L_{req} = 1000B$ and $L_{resp} = 4B$. The results are shown in Fig. 7. The average overhead introduced by the capture mechanism is $\Delta_e = RTT_{baseline} - RTT_{direct} = 0.494\ ms$. The 95% confidence intervals, shown in the figures, are negligible, which testifies that the RTT variability is small.

As a second step, we validate the testbed by verifying that real packets *injected into* the emulation are treated the same way as packets *generated within* the emulation, i.e. they experience the same latency when traversing the emulated 5G network. To accomplish this, we run a simulation where the same request-response application is implemented *within* Simu5G. With reference to Fig. 6, the sender is on the server, whereas the receiver is on one simulated UEs. We compare the RTT measured in this simulated scenario and the one obtained by running the whole testbed with the real application and Host A running Simu5G. The main parameters of the cellular network are shown in Table 1. Fig. 8 and Fig. 9 show the RTT obtained using numerology index $\mu = 0$ and $\mu = 1$, respectively. In the scenario with $\mu = 0$, the RTT differs by $\Delta_{\mu=0} = 0.874\ ms$, whereas for $\mu = 1$, the lag is $\Delta_{\mu=1} = 1.312\ ms$. Clearly, the larger RTT for the emulation is due to packets traversing the emulated network and being captured by the emulation: $\Delta_{\mu=0}$ and $\Delta_{\mu=1}$ are in fact comparable to the $RTT_{baseline}$ measured before. Thus, we can assume that running the emulation does not introduce significant distortions, other than the time required for capturing and injecting packets into Simu5G.

We can now assess the scalability of the testbed with respect to various factors. We first show that the coherence of the emulation is preserved across a large interval of sending rates for the emulated and simulated traffic. We consider UEs (both the real and the simulated ones) receiving 1000B-request packets, initially one per second. We fix $\mu = 0$ (i.e., a TTI of 1ms), the number of simulated UEs to 5 and the number of RBs to 10, whereas we vary, the load of the *real* and *simulated* traffic, and the CQI reporting period. The left chart in Fig. 10 shows that increasing the rate of the traffic injected by the emulated UE does not affect the complexity of the simulation until we get to 1600 kbps, since the RTT values remain constant. The mid chart of Fig. 10, instead, shows that increasing the total offered load of *simulated* traffic increases the RTT when we get to 800kbps. This is because we have multiple simulated UEs in the scenario, which add more complexity to the emulation with respect to having only the emulated UE. The right part of Fig. 10 shows that the CQI reporting period affects the coherence more tangibly: when UEs report their CQI every 10ms, the RTT of the emulated traffic is increased. As a matter of fact, computation of the CQI in Simu5G is a complex task that requires each UE to evaluate the interference of the channel for all the RBs.

To better assess the impact of UEs, we vary the number of simulated UEs and the number of RBs, using a CQI reporting period of 80ms. As shown in Fig. 11, when the number of RBs is small, i.e. 10 RBs, the RTT stays constant while the number of UEs is less than 8, whereas we can use at most 5 UEs when RBs are 25 and at most 3 when RBs are 50. However, we cannot maintain coherence when 100 RBs are used. Results for $\mu = 1$ are shown in Fig. 12. Since $\mu = 1$ means shorter TTIs (i.e. 0.5ms long), the emulation becomes more challenging - Simu5G handles in fact twice as many events per unit of time - and coherence is only preserved for a small number of RBs and simulated UEs.

The above results are promising for at least two reasons. First, they prove that it is actually possible to run a 5G emulation on a desktop machine, in an environment with a gNB and several UEs, whose entire protocol stacks are modeled. We are not aware of similar results in the literature. As far as the number of UEs is concerned, the fact that only few of them can be instantiated in an emulation while maintaining coherence should not be misconstrued as a severe limitation. On one hand, a loss of coherence does not make the emulation worthless: if the RTT stays bounded, the only net effect is that the timing properties of the emulated traffic may not be accurate at the TTI level, but they are still fairly reliable. On the other, the purpose of having a large number of UEs and/or a large overall sending rate in an emulated scenario is usually to saturate a cell, so as to add radio-access delay to the real traffic. The same result can be achieved

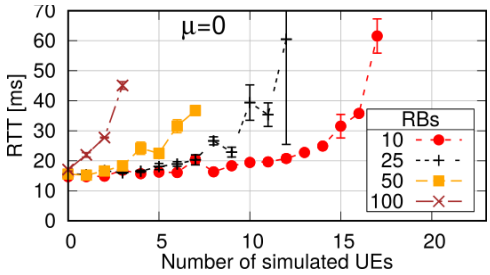

Fig. 11: RTT with increasing simulated UE, $\mu = 0$

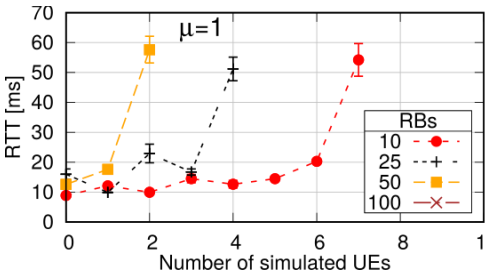

Fig. 12: RTT with increasing simulated UE, $\mu = 1$

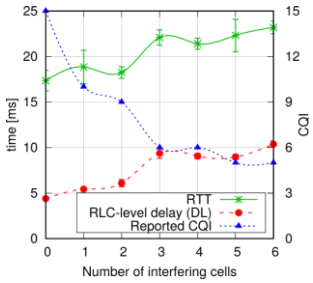

Fig. 13: RTT, RLC-level delay and reported CQI with increasing number of interfering cells

more economically by modeling the *impact on the number of available RBs* of the simulated UEs, without the need of actually including them (or their traffic) in the scenario. All it takes is to modify the scheduler in Simu5G, so that the number of RBs made available to the scheduler on each TTI obeys some custom distribution, which mimics the occupancy of an arbitrary number of simulated UEs. This requires a negligible overhead, irrespective of the number of simulated UEs or their traffic. More to the point, Simu5G already comes with *External Cells* (*ExtCell*) to enable modeling multi-cell scenario at a low complexity. These are simplified gNBs, which do not run the full NR protocol stack, but only occupy a number of RBs of the DL subframe on each TTI, so as to produce inter-cell interference. They enable us to produce a configurable level of interference without incurring the overhead of real gNBs and their served UEs. To show this, we add six interfering cells to the scenario of Fig. 6, deployed on a 500m-radius circumference, centered at the gNB, each occupying the whole available bandwidth. Again, we consider five simulated UEs, 10 RBs and $\mu = 0$. Fig. 13 shows the RTT of the real traffic when activating an increasing number of ExtCells. The RTT does increase with their number, but this is *only* due to the effects of interference on the emulated UE, and not to the emulation being increasingly more complex. In fact, the interference reduces the CQIs, which in turn causes the gNB to use more RBs to serve the same traffic. This means that a single application packet is segmented and transmitted in multiple, subsequent TTIs. This is reflected in the larger delay that packets suffer at the RLC level of the NR protocol stack, whose evolution has the same shape as the RTT curve.

IV. A PROOF-OF-CONCEPT SETUP WITH INTEL OPENNESS

This section describes a proof-of-concept setup of an end-to-end testbed where MEC apps communicate through an emulated 5G network, in a scenario that was demonstrated at the 5GAA workshop in Turin in November 2019 [16]. The MEC hosting is realized using Intel OpenNESS [13]. The latter is an open-source MEC software toolkit that enables highly optimized and performant edge platforms to on-board and manage applications and services with cloud-like agility across any type of network, facilitating development and deployment of the edge platform. Its features include (see Fig. 14):

- *Network complexity abstraction*: it allows any data plane, container network interfaces and access technologies;
- *Cloud-native capabilities*: it supports cloud-native ingredients for resource orchestration, telemetry & service mesh;
- *HW/SW optimizations for best performance and ROI*: it provides node feature discovery and optimal placement of apps/services by exposing underlying edge hardware and enabling control/management of hardware accelerators including dynamic programming, configuration and orchestration.

The main component of the OpenNESS toolkit is the Edge Host, which implements the whole functionalities of a MEC Host, e.g. MEC platform, traffic steering etc., and includes a virtualization infrastructure for running the MEC Apps. The latter are Docker containers that are installed to the MEC Host and started via the GUI provided by the OpenNESS Controller, which acts as MEC Orchestrator.

The high-level representation of the implemented testbed is shown in Fig. 15 and it is composed of a client, a server running the OpenNESS software and the Simu5G emulated network in between. The considered application involves the client requesting a video-stream, hosted by the MEC Host, whose video quality can be changed dynamically, e.g. according to the quality of the radio channel in the emulated network. The MEC App is a video streaming application that applies real-time transcoding to a video file provided as input, i.e. converts "on the fly" the video to H264 format using the *x264* encoder, and makes the output of the transcoding process available for network streaming via HTTP on the well-known TCP port 8080. We do this using the open-source VideoLan Converter (VLC) software v4.0 [14]. We modified the *x264* module within the VLC package so that the *average* bitrate[2] of the video can be changed when notified by an external source. In our implementation, VLC receives the trigger via *telnet* commands. This allows us to change the bitrate dynamically, e.g. according to the condition of the underlying network, emulated through Simu5G. We compiled and installed the modified version of the VLC software on a Linux Ubuntu 18.04 Docker container and uploaded it to the OpenNESS Edge Host. When the Docker is started, it streams an excerpt of the 720p version of the "Big Buck Bunny" movie [15], whose average bitrate is about 1.4 Mbps. On the client side, the original VLC player is used for streaming the video from the network.

We emulate the same single-cell scenario in Fig. 6. The UE is located 50m far from the gNB, i.e., having a good channel quality in absence of interference. The available bandwidth is limited to five RBs, so as to reach a saturation condition quickly. To observe the effects of varying network conditions on the

[2] Since a video is inherently VBR, we cannot instruct VLC to set a constant or capped bitrate.

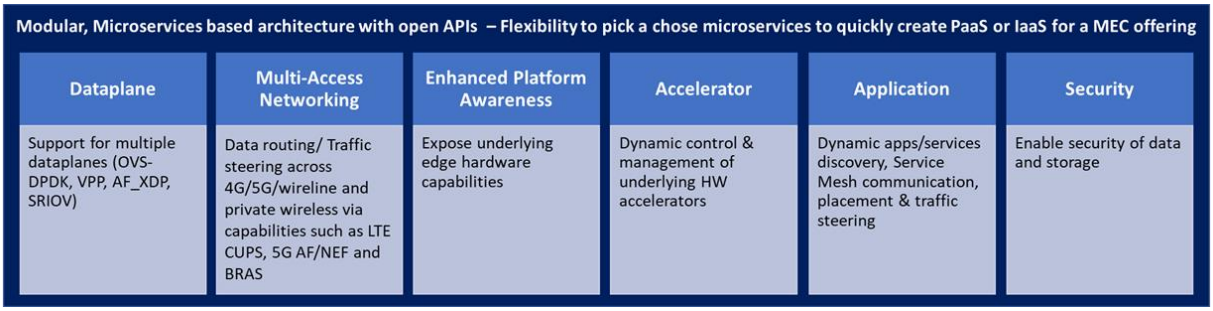

Fig. 14: OpenNESS capabilities

quality of the video stream, we use one ExtCell located 70m far from the UE. The ExtCell starts occupying 100% of the bandwidth after 25 seconds of simulation. In this scenario, the CQI reported by the UE instantaneously drops from 15 to 6, increasing the number of required RBs to satisfy the traffic. Initially, the video is streamed at its maximum bitrate. When the ExtCell starts generating interference at $t = 25s$ and the CQI drops to 6, the air frame fills, hence larger delays occur and the video at the client side presents impairments and interruptions. For this reason, before reaching the network saturation, we can send a command to the MEC app to reduce the bitrate. This way, the video goes on without interruptions, albeit at a lower quality, occupying fewer RBs. Fig. 16 shows the quality of the video before and after reducing the bitrate. The above proof of concept shows that Simu5G is a powerful tool for testing services like MEC-assisted video streaming, allowing one to assess the effects of a realistic network environment on the application.

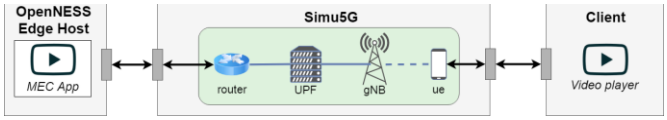

Fig. 15: Proof-of-concept setup

V. Conclusions and Future Work

In this paper, we have presented the real-time emulation capabilities of Simu5G, a novel 5G New Radio simulator based on OMNeT++ and INET. We have shown how to setup a testbed where real applications communicate through an emulated 5G network. We have validated the testbed and profiled it to assess its scalability with respect to various factors. We found that the limiting factor seems to be the number of internally simulated UEs and their CQI reporting period, due to the interference computation, which has to be repeated as many times as the UEs. This is not a big limitation, since the effects of *many* UEs on the air-frame occupancy can be modeled *without* modeling the UE themselves. We have also reported about the use of Simu5G to demonstrate MEC apps, in conjunction with the Intel OpenNESS toolkit, based on a recent demonstration in 5GAA.

At the time of writing, we are actively pursuing the following extensions to this work: re-factoring the physical layer in Simu5G to take advantage of GPU parallel computation; adding MEC services *within* Simu5G, e.g. location and radio-network information or, so that a MEC app running (e.g.) on OpenNESS could leverage the latter to build advanced services.

Acknowledgments

Work partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence). The subject matter of this paper includes description of results of a joint research project carried out by Intel Corporation and the University of Pisa. Intel Corporation reserves all proprietary rights in any process, procedure, algorithm, article of manufacture, or other results of said project herein described.

References

[1] A. Virdis, G. Stea, G. Nardini, "Simulating LTE/LTE-Advanced Networks with SimuLTE", DOI 10.1007/978-3-319-26470-7_5, in Advances in Intelligent Systems and Computing, Vol. 402, pp. 83-105, Springer, ISBN 978-3-319-26469-1, 15 January 2016.

[2] SimuLTE Website. http://simulte.com, accessed April 2020

[3] G. Nardini, G. Stea, A.Virdis, D. Sabella, "Simu5G: a system-level simulator for 5G networks", Simultech 2020, online conf., 8-10 July 2020

[4] Simu5g website, http://simu5g.org

[5] OMNeT++ Website: https://omnetpp.org, accessed April 2020.

[6] INET Library Website. https://inet.omnetpp.org, accessed April 2020.

[7] G. Nardini, A. Virdis, G. Stea, "Modeling network-controlled device-to-device communications in SimuLTE", MDPI Sensors, 18(10), 3551, DOI: 10.3390/s18103551, 2018

[8] C. Mehlfuerer, *et al.*, "Simulating the long term evolution physical layer", in Proc. 17th EUSIPCO, Glasgow, UK, 2009

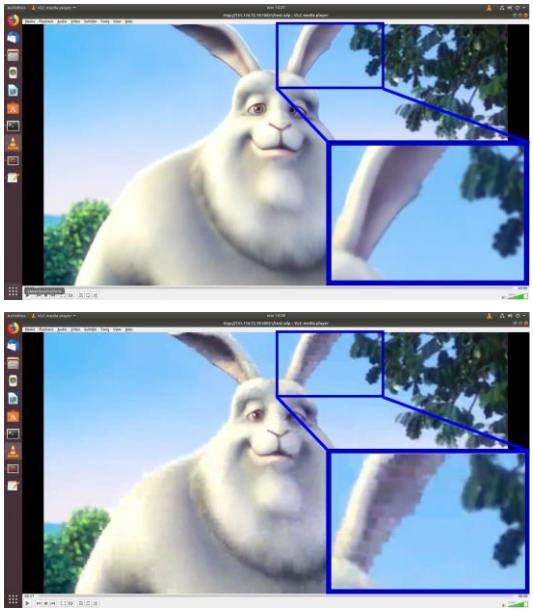

Fig. 16: Quality of the received video with high (top) and low (bottom) bitrate

[9] C. Sommer *et al.*, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," IEEE Transactions on Mobile Computing (TMC), vol. 10 (1), pp. 3-15, January 2011

[10] B. Sliwa, C. Wietfeld, "LIMoSim: A Framework for Lightweight Simulation of Vehicular Mobility in Intelligent Transportation Systems", in: A. Virdis, M. Kirsche, (eds.) "Recent Advances in Network Simulation", Springer, Cham, pp. 183-214, May 2019

[11] https://www.tcpdump.org/

[12] 3GPP TR 36.873 v12.7.0, "Study on 3D channel model for LTE", Dec. 2017

[13] https://www.openness.org/

[14] https://www.videolan.org/

[15] https://peach.blender.org/

[16] https://5gaa.org/news/5gaa-live-demos-show-c-v2x-as-a-market-reality/