Received October 16, 2021, accepted October 25, 2021, date of publication October 27, 2021, date of current version November 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3123873

Scalable Real-Time Emulation of 5G Networks With Simu5G

GIOVANNI NARDINI, GIOVANNI STEA, AND ANTONIO VIRDIS
Dipartimento di Ingegneria dell'Informazione, University of Pisa, 56122 Pisa, Italy
Corresponding author: Giovanni Stea (giovanni.stea@unipi.it)

This work was supported in part by the Italian Ministry of Education and Research (MIUR) in the framework of the Cross-Lab Project (Departments of Excellence), and in part by the European Commission through the H2020 Projects Hexa-X under Agreement 101015956.

ABSTRACT Real-time emulation of 5G networks is highly beneficial for several purposes, such as prototyping or performance evaluation of distributed applications meant to run on 5G networks, research demonstration, evaluation of other technologies (e.g., Multi-access Edge Computing) meant to interoperate with 5G access. In this work, we describe how to use Simu5G, a new end-to-end simulator of 5G networks based on OMNeT++, as a real-time emulator. We describe in detail the modeling choices that allow emulation to scale up without compromising accuracy. We present a thorough evaluation of the Simu5G's emulation capabilities, showing that networks with hundreds of simulated users and tens of cells can be emulated on a single desktop machine.

INDEX TERMS Emulation, real-time, Simu5G, simulation.

I. INTRODUCTION

Fifth-generation (5G) cellular networks will bring significant changes to the wireless networking landscape. In fact, they will enable unprecedented scenarios, such as smart cities, autonomous vehicles, augmented reality and Industry 4.0. Most of these services will be composed of both *communication* and *computation*, leveraging the deployment of computing and storage capabilities in the cloud-edge continuum. 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 [30].

Application 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. During the application development phase (hence before the application is brought onto the market and run on the live network), developers would benefit from knowing what performance they should expect from a 5G network in terms of bandwidth and latency, at the very least, so that they can in turn adapt the behavior and/or requirements of the application. On the other hand, 5G infrastructure owners and operators 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 applications exchange traffic through a 5G network. Unfortunately, 5G network testbeds are hard to come by, especially for developers.

We recently developed Simu5G [3], [4], a discrete-event simulation library for 5G New Radio networks based on OMNeT++, which evolves from the well-known SimuLTE simulator of LTE/LTE-A networks [2] and is backward compatible with it. Simu5G is already being used by a large community of researchers. It allows one to setup arbitrarily complex network scenarios involving 5G access, leveraging network models taken from the INET library [6], vehicular mobility taken from Veins [12] or Artery [13], etc. This makes Simu5G capable of simulating scenarios including the full Internet protocol stack in order to evaluate the performance of applications exploiting 5G connectivity, as well as how network-level metrics are affected by the traffic generated

The associate editor coordinating the review of this manuscript and approving it for publication was Jesus Felez.

by applications. An application developer can use Simu5G in at least three different – and complementary – ways: first, by building a *model* of its application, to the required level of abstraction, and integrating the latter in Simu5G; second, by compiling the *real codebase* with Simu5G, substituting network calls with few localized modifications. These are both useful for functional testing and non-real-time performance evaluation (e.g., message counting). We show in this paper that a third way exists, i.e., to use Simu5G as network transport, having application endpoints exchange packets through it, *in real time*. In the literature (e.g., [31]–[33]), such approach is referred to as *emulation* – and we will do the same from now on – since packets exchanged by real applications with the simulator perceive the same impairments (e.g., delay and losses) as if they were running on the real network. This is useful to test the real-time performance of an application, e.g., when user input is required, or when closed-loop sensing and control applications are to be tested. 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 preproduction environment, so as to obtain confidence regarding their performance. The main benefit of using Simu5G as a real-time emulator is that users can rapidly define a custom, arbitrarily complex 5G network scenario and run their own application through it, in order to measure its performance or demonstrate it live to a third party, without buying expensive hardware to implement a real network testbed. For instance, developers of a multiplayer virtual reality game may be interested in testing whether the latency introduced by certain 5G network configurations could be detrimental for the Quality of Experience perceived by the users of their application. Simu5G allows them to assess such performance in a simple and cost-effective way. Note that using a real 5G network (e.g., a commercial one provided by a telecommunications operator) would allow users no control on the experimental conditions. Using Simu5G as an emulator can also be beneficial to network operators, e.g., to test the impact of real applications (instead of models thereof) on their network.

Real-time emulation with Simu5G is possible thanks to the combination of several concurring factors. First, the capabilities of the OMNeT++ environment to run a real-time event scheduler, and of the INET library to exchange packets between a running simulation and the operating system. Second, but certainly no less important, a careful modeling of Simu5G functionalities and entities is required to make the software *lightweight enough* to be executed in real-time on a desktop machine. In fact, real-time event scheduling is only possible if simulator events are processed faster than wall-clock time on the host machine. This means that, depending on the network model, there will be a maximum *scale*, in terms of number of network entities (e.g., User Equipments and Base Stations), carried traffic, etc., beyond which real-time emulation will be impossible on a given host machine.

This paper presents the architectural and modeling choices that allow Simu5G to run in real time up to fairly large scales (i.e., ten cells, a thousand UEs). A scale such as this is required to create scenarios where realistic intercell interference and resource contentions are emulated. Our solution is to provide *multiple* models of the main network elements (i.e., UEs and cells), representing different tradeoff points between the level of detail and the processing overhead. For instance, a standard UE model is endowed with the entire protocol stack up to the application layer, and it can be used as an application endpoint during emulation and/or to compute node-specific or end-to-end metrics. A *background* UE, instead, only needs to generate uplink (UL) interference and resource contention at the MAC layer, without recording any kind of UE-specific metric. This makes it possible to do away with the processing overhead of all the upper layers, while still enabling it to participate to uplink/downlink MAC scheduling, occupy resources, and generate interference. A similar approach is used for cells: a *background* cell generates downlink (DL) interference correctly without the overhead of simulating the full model of their associated UEs. Moreover, we report a thorough performance evaluation of Simu5G's emulation capabilities, identifying the limits in all the relevant factors (number of simulated UEs and cells, application traffic, system bandwidth, number of carriers, numerologies). Our results show that Simu5G can run in real time, carrying traffic up to 1-2 Mbps between external endpoints, while emulating a scenario with ten 5G cells and a thousand users. To the best of our knowledge, this is the first work documenting a free, open-source simulator featuring 3GPP-compliant 5G New Radio implementation that can run as both a simulator and a real-time emulator with the *same* codebase, at that scale. Being able to use the same codebase for both simulations and live prototyping abates the developing time and makes results more reliable and easier to demonstrate. For instance, Simu5G is being used in the framework of the Hexa-X EU project [8], to support the development and demonstration of federated learning algorithms for Explainable Artificial Intelligence (XAI) models.

The rest of the paper is organized as follows: Section II introduces the necessary background, describing the basic functionalities of OMNeT++ and Simu5G. Section III discusses the modeling choices that allow scalable real-time emulation. Section IV evaluates the emulation capabilities of Simu5G on a standard desktop computer, while Section V reviews the related work. Finally, Section VI draws conclusions and highlights directions for future work.

II. BACKGROUND

This section introduces the necessary background. More specifically, we briefly describe the OMNeT++ framework and the INET library, focusing on the features connected with real-time emulation, and then we present an overview of Simu5G.

A. THE OMNeT++ FRAMEWORK AND THE INET MODEL LIBRARY

OMNeT++ is an open-source framework for discrete-event simulation. It is free for academic use, and it has a very large users' community (including corporate ones). OMNeT++ is built on the concept of *module*. OMNeT++ modules exchange *messages* through *connections* linking their *gates*. A simulation model is represented by a *network*, which is a top-level compound module, with no external gates. Hierarchy must be preserved when connecting modules: with reference to FIGURE 1, simple module 1 can only connect to the compound module gate, and not to 2 directly. This forces a user to design models that are easier to verify and extend. *Event handlers*, called by the simulation kernel on receipt of messages, implement the behavior of a module. For instance, a module can schedule a timer by sending a message to itself, set to be handled at the expiration time. Simple modules have an *initialization* and *finalization* function, which are called at the start and the end of a simulation. The OMNeT++ framework has built-in functionalities, such as event queueing or random number generation, thus allowing users to concentrate on writing their own simulation models.

OMNeT++ maintains separate model's *behavior, description* and *parameter values*. Object-oriented C++ is used to code a model's behavior. The description (i.e., gates, connections and parameter definition) is instead coded separately using the Network Description (NED) language. NED is a declarative language, which exploits inheritance and interfaces, and it is fully convertible into XML. Using NED; one can write *parametric* simulation scenarios, e.g., a hub-and-spoke network with a variable number of spokes. Last, but not least, parameter values are written in Initialization (INI) files. INI files are read at runtime, and initialize the model. Multiple values or intervals can be specified for a parameter, which makes it easy to run simulation studies where factors vary.

In OMNeT++, simulated 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 scenario being run. Typically, there will be a *scale* in terms of number of UEs, gNBs, or traffic transmitted within a 5G network, beyond which a given simulation will not be able to run in real time.

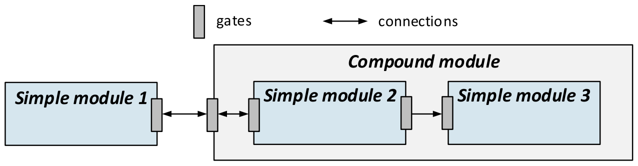

FIGURE 1. OMNeT++ modules and their connection.

INET[1] is a popular model library devised for OMNeT++. It includes a wealth of models of computer network elements, e.g., hosts, protocols, routers/switches, connections, etc.. Notably, INET models many protocols of the TCP/IP suite, such as TCP, UDP, IPv4, IPv6, OSPF, BGP, etc., as well as wired and wireless layer-2 protocols (Ethernet, PPP, IEEE 802.11, etc.). A user can incorporate the INET library into an OMNeT++ simulation scenario, and then be able to run a simulation where an application, residing on a host, communicates with another endpoint through a network of IP routers, each having Ethernet cards and PPP WAN connections. The two endpoints themselves may be using different access technologies, e.g., 802.11 and 802.15, without this being a problem.

Moreover, the INET library comes with modules that interface the simulation environment with the host operating system (OS). They make packets received by the host OS from either real network interfaces or local applications appear in the simulation, and send packets generated within the simulation out to the OS, that will process them as any other network packet, i.e., forward them to either real network interfaces or local applications. To do this, two conditions must apply for the simulated modules that send/receive packets from/to the host OS. First, the IP addresses they are configured with in the simulation must be known to the host OS so that it can forward packets to them, i.e., they must be "routable" within the host OS. This is achieved by adding dedicated IP routes to the routing table of the latter. Second, such simulated modules must be equipped with a new submodule provided by the INET library, called *ExtInterface*. The INET library provides specialized implementations for the *ExtInterface* module, each of them allowing the simulation to exchange packets at different layers of the network protocol stack, e.g., sending/receiving IP datagrams or UDP segments. For the scope of this paper, we only describe the *ExtLowerEthernetInterface*, which exchanges Ethernet frames with the host OS using raw sockets. For incoming packets, an emulation-enabled real-time scheduler is responsible for fetching such packets from the socket, to convert them into the equivalent C++ object representation used in the simulation, and to add them to the event queue, where they are processed alongside the other events generated within the simulation. However, the real-time scheduler fetches new packets from the socket *only* when the simulation time is in line (henceforth, *coherent*)

[1]In this paper, we refer to version 4.3 of the INET library.

with wall-clock time. When the simulation time lags behind the wall-clock time, real packets stay in the socket buffer and accumulate delay, until coherence is resumed. Moreover, if real packets arrive faster than the rate at which the scheduler drains the socket buffer, the latter fills up and new packets are eventually discarded. When packets need to be sent outside the simulation environment, they are transmitted to the host OS using the same socket. The above features allow one to use Simu5G as a network emulator, that transports packets of real applications, delaying them the way a 5G network would.

B. OVERVIEW OF Simu5G

Simu5G [3], [4] is based on OMNeT++ and it incorporates models from the INET library. It represents the evolution of the well-known SimuLTE 4G network simulator [2] towards 5G NewRadio access, and incorporates all SimuLTE's functionalities, allowing one to evaluate legacy or mixed scenarios as well. Simu5G simulates the data plane of both the core and the radio access networks.

As far as the core network is concerned, it allows users to instantiate a User Plane Function (UPF) or Packet Data Network 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 connected to the core network directly, as shown in FIGURE 2 (left), in the so-called standalone deployment. Alternatively, a gNB can operate in an E-UTRA/NR Dual Connectivity (ENDC) deployment, shown in FIGURE 2 (right), where LTE and 5G coexist [9]. 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 for an LTE eNB, which acts as master node connected to the core network. The eNB and the gNB are connected through the X2 interface and all NR traffic traverses the eNB first.

UEs and gNBs are modeled as compound OMNeT++ modules. Their architecture is shown in FIGURE 3. UEs have all the protocol stack until the application layer, whereas gNBs only have layer-3 functionalities. Both include a NR Network Interface Card (NIC), which models the NR protocol stack. The structure of the NR NIC modules is shown in FIGURE 3, and is different for the two entities. Notably, UEs have a dual stack (both LTE and NR), which allows them to leverage double (4G/5G) connectivity in an ENDC scenario, in-sequence packet delivery being guaranteed by the Packet Data Convergence Protocol. Packet transmission entails top-down traversal of the NR protocol stack shown in FIGURE 3, with messages exchanged by neighboring modules. Conversely, packet reception entails bottom-up traversal. Note that OMNeT++ messages are *events*: the price to pay for complete modeling of the layers within the NR protocol stack is that the transmission of a single IP packet via the NR interface requires Simu5G to handle a sizable number of events,

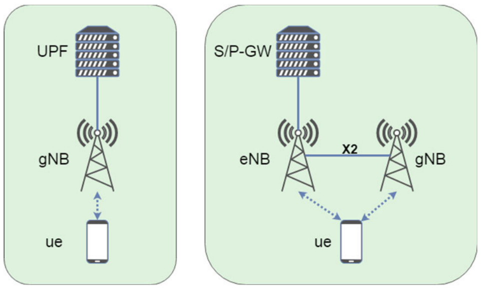

FIGURE 2. Standalone (left) and ENDC (right) deployment.

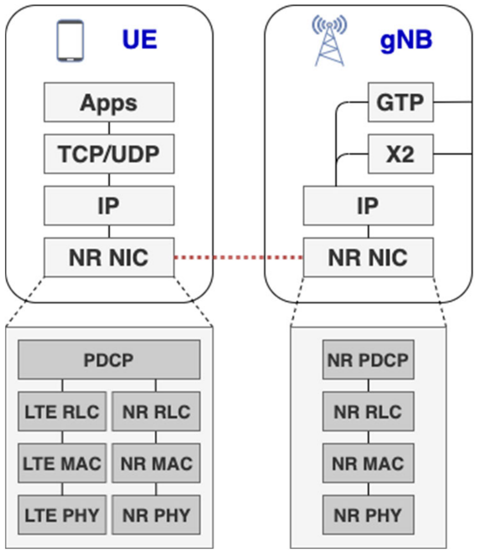

FIGURE 3. Simu5G main modules.

in the order or few tens, among inter-layer communication, fragmentation/reassembly, timers, ACK/NACK sending, etc..

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 Protocol Data Unit (PDU) to a receiver, the PHY modules of the two entities exchange an OMNeT++ message, whose propagation delay is set to the duration of one NR time slot. 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 $P_x = BLER(CQI, SINR_x)$, the error probability for that RB given the Channel Quality Indicator (CQI) used by the sender and the received SINR.

This is done by using Block Error Rate (BLER) curves, which can be obtained from measurements performed using link-level simulators (e.g., [11]). With reference to FIGURE 4, BLER curves have the SINR as an abscissa, an error probability as an ordinate. There is one curve for each CQI, and the one corresponding to the transmission format used by the sender is selected. Then, error probability P_x is found at the abscissa represented by $SINR_x$;

- 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 [10] 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). Carrier Aggregation is supported, and different component carriers (CCs) can be configured with different numerologies and either FDD or configurable TDD slot format. Moreover, different CCs can have different channel models. Simu5G also 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 [12] or LIMoSim [13]).

III. SCALABLE MODELING OF UEs AND CELLS IN Simu5G

In this section we explain the rationale behind our modeling choices, and then describe the different models of UEs and cells that can be instantiated in Simu5G. In the next Section, we will present a validation of the models, comparing the processing time and accuracy when using each version of the entities.

A. MOTIVATION AND ANALYSIS

We first introduce a motivating example. It is often the case that the focus of a simulation/emulation experiment is on endpoints located in a single cell, or few cells at most. With reference to FIGURE 5, one may want to measure the latency and jitter of a D2D offloading scheme. UE 1 receives a Constant Bit Rate stream from its serving gNB, and relays it to UE 2 using D2D resources allocated on the UL spectrum. Both UEs move (while remaining close to each other), and - at some point - traverse cell borders, changing their serving gNB. This scenario can be simulated with Simu5G. We discuss what exactly should be simulated, in order for it to be realistic. Obviously enough, we need at least two cells (i.e., two gNBs) and two UEs. However, if we limit ourselves to this, communications (both the gNB-1 DL leg, and the 1-2 D2D leg) will have no interference, and no resource contention will take place anywhere, making for

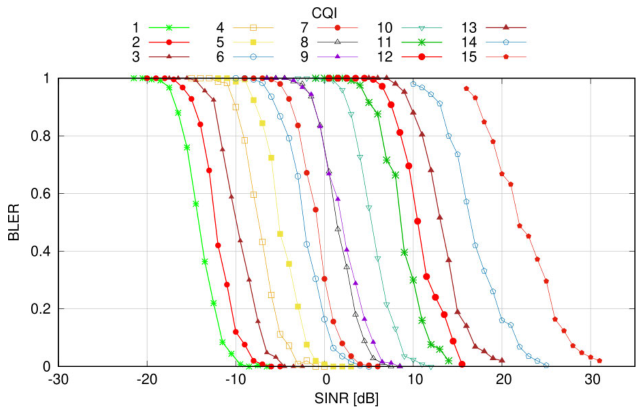

FIGURE 4. BLER curves employed in Simu5G.

a very optimistic assessment of latency and jitter. As an aside, we observe that a bog-standard desktop computer would have little trouble emulating *that* scenario in real-time. Should one want to introduce resource contention on both the DL and the D2D legs, one should *increase the number of UEs*, and their associated application traffic to be ultimately transmitted via the DL and UL (or D2D). Moreover, and worse yet, in order to have realistic interference levels (in both the UL and the DL), one should add to this scenario other neighboring cells (say, a whole tier at least), each one having its own set of UEs and associated application traffic. The result would then be a very realistic scenario, with a number of UEs in the hundreds or thousands and a number of cells in the few tens. However, its required processing load per unit of time would put real-time emulation out of the picture.

More in detail, three processes within Simu5G are particularly computation-intensive: *MAC-level scheduling* at the gNBs; *PHY-layer reporting* at the UEs, and *protocol stack traversal* for a packet. Assume that we are running a scenario with B gNBs, N UEs per gNB, and C carriers per gNB. MAC-level scheduling needs to be performed on each TTI (hence, up to 16 times per ms, depending on the numerology) and separately for each CC and it involves sorting all the UEs in a cell according to some metric. This implies $O(B \cdot C \cdot N \cdot logN)$ operations per TTI ($N \cdot logN$ being the cost of sorting N UEs). PHY-layer reporting is done by a UE at a configurable reporting interval (e.g., 10 TTIs) independently for each CC, and involves measuring the signal strength received by all the gNBs it can hear (i.e., running $O(B \cdot C)$ instances of channel models, involving floating-point algebra and transcendent functions), compute the SINR, and send a message with the reported CQIs to the serving gNB. This entails an overall $O(B^2 \cdot C \cdot N)$ operations and $O(B \cdot N)$ OMNeT++ messages (hence events to be handled) per reporting interval. As far as protocol stack traversal is concerned, we have already mentioned that a single IP packet generates tens of events in the system, due to the layered nature of the NR protocol stack. More operations are required to run layer-4 protocols, especially TCP. In this case the load depends not so much on the number of entities, rather on the generated traffic (hence, ultimately, on the system capacity and scale).

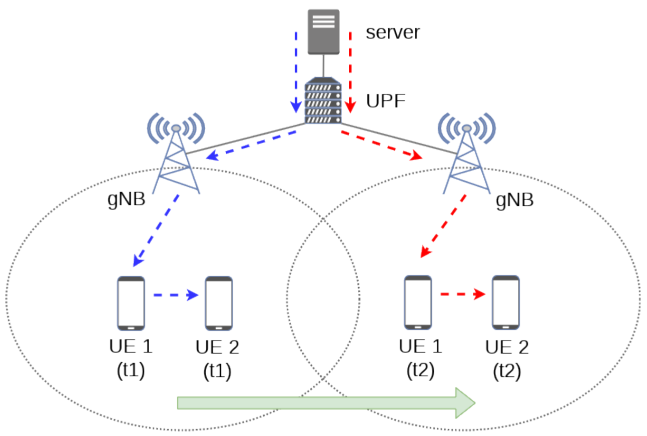

FIGURE 5. Example scenario.

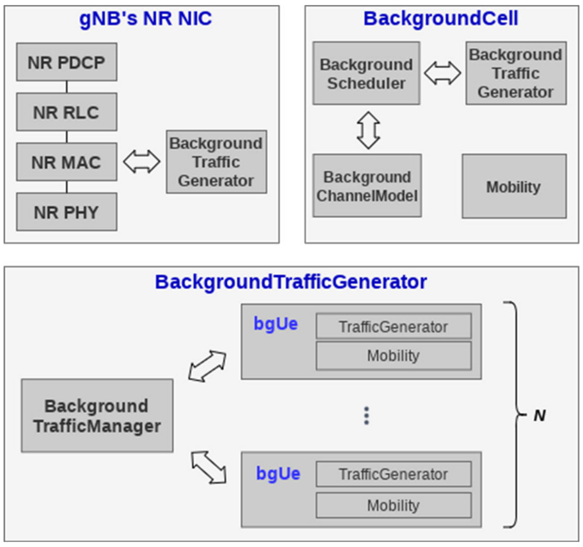

FIGURE 6. Modelling of i) the enhanced gNB's NR NIC module (Foreground cell), ii) the *BackgroundCell* module, and iii) the *BackgroundTrafficGenerator* module.

Parallel/distributed simulation is often used to address scalability issues such as these. However, parallelism works best when processing-heavy, low-interaction models are distributed across machines, which is not the case here: should one distribute *cells* across computing cores, global interference computation would still require per-TTI synchronization across cells. Moreover, parallel emulation has its own challenges, such as maintaining real-time synchronization across instances, with sub-ms accuracy, and it does require a non-negligible amount of hardware. In this paper, we choose a different path: we provide alternative *lightweight* models of UEs and cells, with limited functionalities, to be used for the sole purpose of creating resource contention and interference. This allows users to instantiate a judicious mix of *complete* and *lightweight* versions of the entities, so as to create a realistic scenario while keeping the processing load to a minimum. We describe the lightweight models of UEs and cells, and validate them by comparing scenarios with the two versions of each model.

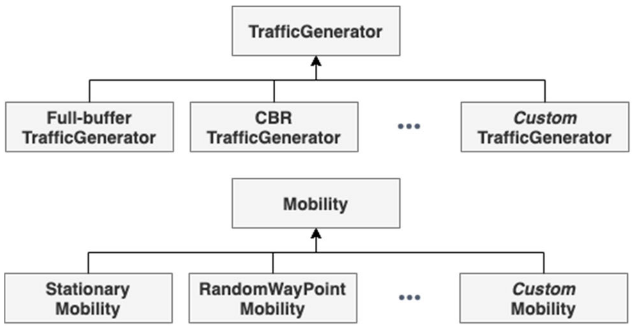

FIGURE 7. Inheritance of TrafficGenerator and mobility modules.

B. MODELING

We distinguish between *foreground (FG)* and *background (BG)* UEs, depending on their role within a simulation experiment. A FG UE is one whose application packets and/or statistics you will want to observe. In a FG UE, all the protocols must be modeled (from the application to the PHY sublayer of the NR stack), and PHY-layer reporting must be enabled. On the other hand, a BG UE needs not have the full protocol stack and exists only for the sake of creating resource contention at the MAC level and interference. Thus, a BG UE basically consists of a *traffic generator* and a *mobility model*.

Similarly, we distinguish FG and BG *cells*. A FG cell is one to which at least one FG UE is associated, whereas a BG cell only serves BG UEs. BG cells are employed to create interference to FG cells. With reference to the example in Section III.A, the only FG cells are those in which UEs 1 and 2 move about, whereas all other cells are BG ones. Moreover, the only FG UEs are 1 and 2, whereas all the others (even in FG cells) are BG UEs. A FG cell needs to perform all the operations required by all the layers of the protocol stack including scheduling in *both* the DL and the UL directions. This is because DL/UL packets sent to/received from its FG UEs need to traverse the air interface, hence they must be correctly delayed. On the other hand, a BG cell does not need to implement the whole protocol stack, and DL scheduling is not required. In fact, in order to generate interference on nearby cells, it is only necessary to know *how many RBs* are occupied on each TTI in the DL (as opposed to *which BG UE occupies which RBs*, which would instead require scheduling). UL scheduling is instead required in BG cells: in fact, in the UL, interference is generated by BG UEs, on specific RBs at specific TTIs, depending on UE's traffic and CQIs. Hence, we do need UL scheduling to determine *which BG UE occupies which RBs.*

Since the only tangible effect of BG UEs in both FG and BG's MAC layers is to affect the scheduling process at the gNB itself, we choose to deploy them using a new module called *BackgroundTrafficGenerator* within both FG and BG cells.

FIGURE 6 depicts the enhanced internal architecture of the NR NIC of a FG cell and the new *BackgroundCell* model for the BG cell, both including a *BackgroundTrafficGener-*

ator submodule. The latter is shown in the bottom part of FIGURE 6 and includes *i*) a vector of modules representing a configurable number of BG UEs, and *ii)* a *BackgroundTrafficManager* module that manages the interaction between the BG UEs and the gNB's protocol stack. In turn, each BG UE is composed of two submodules, namely *Mobility* and *TrafficGenerator*. The former stores the position of the BG UE in the network floorplan and manages its movement, whereas the latter stores the size of the BG UE's data buffer according to some traffic generation pattern. Both *Mobility* and *TrafficGenerator* can be extended through inheritance of OMNeT++ modules to configure the desired mobility pattern and traffic generation distribution, as shown in FIGURE 7. Note that a *TrafficGenerator* has two buffers, one for the UL and one for the DL, allowing asymmetric traffic in the two directions. At every scheduling period, the gNB gets the information about backlogged BG UEs from the *BackgroundTrafficManager* and schedules them together with FG UEs, if any, consuming bytes from their data buffer and allocating RBs accordingly. Despite being simpler, the above modeling for BG UEs allows the same degree of flexibility as using FG UEs when building simulation scenarios: one can simulate multiple gNBs, each serving a configurable number of BG UEs and, in turn, each BG UE has its own mobility and traffic generation pattern. As a side note, we observe that this modeling allows one to easily create *full-buffer* UEs, which are often employed in 3GPP reference scenarios [14]: a full-buffer UE is a BG UE whose *TrafficGenerator* replenishes DL/UL buffers every TTI.

In order to allocate RBs to BG UEs, the gNB's scheduler must know the CQI of the latter. However, we discussed above that accurate CQI reporting requires computation-intensive PHY-layer operations, which hampers the scalability of emulation. For this reason, we allow users to select between two methods to obtain the CQI of BG UEs: *realistic* and *probabilistic*. These represent different trade-off points between accuracy and computation overhead. Realistic CQI computation and reporting is the same employed in FG UEs. It entails computing CQIs including the effects of mobility and interference, using channel models to compute signal attenuation and computing the SINR on each occupied RB, hence it is costly. With probabilistic CQI computation, instead, each BG UE is assigned a *mean* and a *variance* for its CQI: during the simulation and at every CQI reporting period, BG UEs extract a random value for the CQI, assuming a set distribution (the Normal one being the default). On one hand, this makes the CQI computation fast, as it only requires generating a random variate. On the other hand, probabilistic CQI computation is only viable if UEs are stationary: if they are mobile, their position and the interference they perceive will change over time, and so should the probability distribution of their CQI. We allow a user to choose the style of CQI reporting dynamically, i.e., while the simulation is ongoing. For instance, in a scenario when a single FG UE moves from cell 1 to cell 2, BG UEs in cell 1 may switch from realistic to probabilistic, while BG UEs in cell 2 may instead switch from probabilistic to realistic, right at the time of the handover.

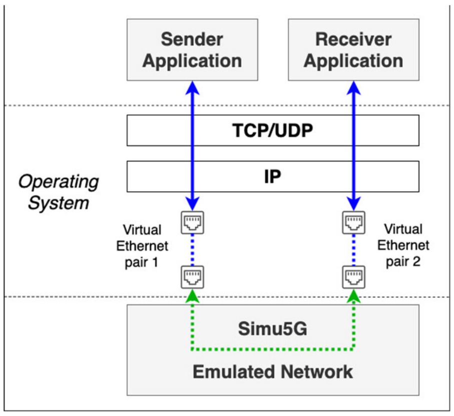

FIGURE 8. Testbed setup.

As far as the *BackgroundCell* module is concerned, the *BackgroundTrafficGenerator* submodule described above represents BG UEs under the control of such BG cell. Then, a module called *BackgroundScheduler* takes care of allocating RBs in both UL and DL directions. To accomplish that, it interacts with the *BackgroundTrafficGenerator* to get the BG UEs' buffers and stores data structures relevant to the resource allocation. Moreover, the *BackgroundCellChannelModel* deals with the computation of SINR for the BG UEs according to the configurable parameters provided by this module (e.g., carrier frequency, transmission power and so on), whereas the *Mobility* module stores the position of the BG cell in the floorplan. Using this model, one can deploy the desired number of BG cells in the floorplan and each of them can be configured with its own channel model parameters and BG UEs.

In an emulation experiment, we can capitalize the above modeling to increase the scale of network scenarios considerably. In fact, the UEs attached to external interfaces need be FG UEs, whereas the other UEs can be modeled as BG UEs. Similarly, only the cell(s) traversed by the emulated UEs need to be FG cells, and all the others can be BG cells. Consider, for instance, a scenario where an emulated UE models a car traversing urban roads, performing handover every now and then. The roads host other cars, and there are residential 5G users around. All UEs except the emulated car are BG UEs. There needs to be only one FG cell at any time (the one hosting the emulated car). Only cars have realistic CQI computation, whereas residential users have probabilistic CQI computation. Possibly, cars in faraway cells can be dynamically stopped (i.e., their position is not updated) and switched to probabilistic CQI computation as well to further decrease complexity, as this will have negligible impact on the FG cell. We will show in the next section that the modelling described so far allows one to emulate scenarios with ten cells and up to one thousand UEs on a desktop PC, with negligible (or null) loss of accuracy.

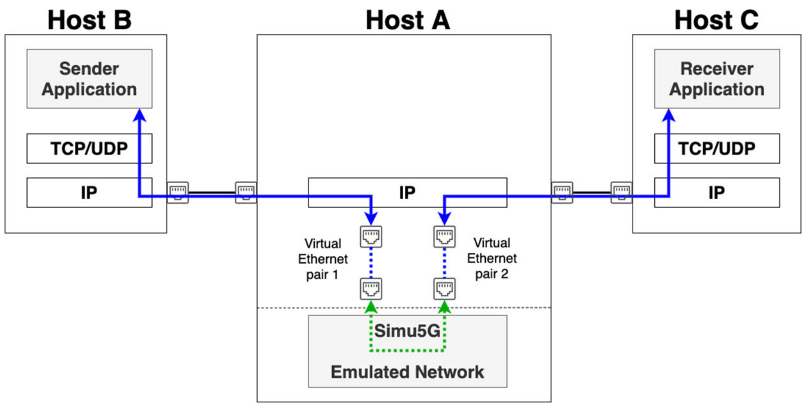

FIGURE 9. Distributed testbed.

IV. PERFORMANCE EVALUATION 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 (e.g., due to CQI reporting). In this section, we evaluate the performance of Simu5G emulation, with the aim to identify which factors constrain 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 that runs a request-response application composed of two processes, namely the sender and the receiver. Packets are forwarded through an intermediate process 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 application 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 Round-trip Time (RTT) we can have an indirect coherence assessment: when the emulation is not coherent, the number of transmitted

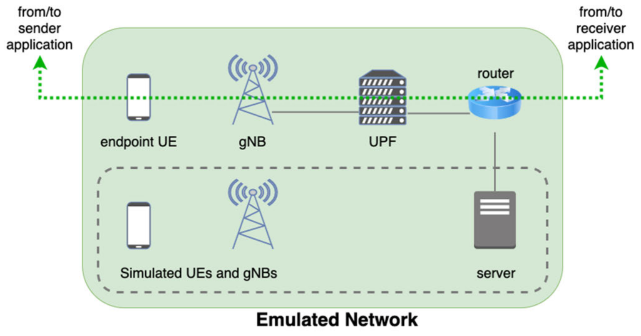

FIGURE 10. Emulated network scenario.

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, which makes this method quite reliable. Throughout this section, we take care to distinguish *simulated time* and *real time* whenever confusion may arise.

A. EXPERIMENTAL CONFIGURATION

Our testbed, shown in FIGURE 8, is composed of one general-purpose 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. The host installs OMNeT++ version 6.0 and the INET library version 4.3. In order to reduce the emulation overhead as much as possible, we built Simu5G using the OMNeT++'s *release* mode, which performs additional code optimizations to the OMNeT++ simulation kernel during compiling, and we disabled the reporting of the Simu5G metrics, since we are only interested in application performance in this testbed. We run the Simu5G network emulation and one real application that generates the real network traffic to be sent through the Simu5G network emulation. The application is composed of two processes, i.e.,

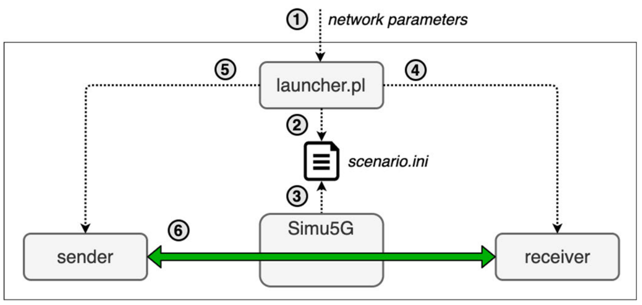

FIGURE 11. Launching the testbed.

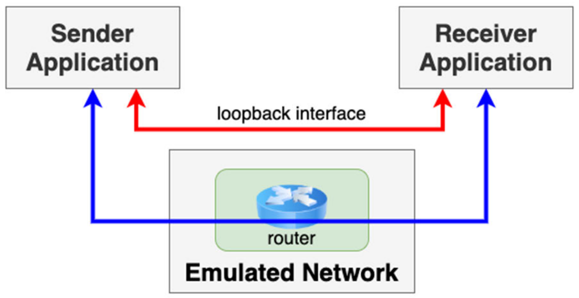

FIGURE 12. Baseline scenario.

TABLE 1. Main network parameters.

Parameter Name	Value
Carrier frequency	2 GHz
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	40 TTIs
Path loss model	[15], Urban Macro (UMa)
Fading model	Jakes
Shadowing model	Log-normal distribution
UE mobility	Static
Traffic type	Req-resp

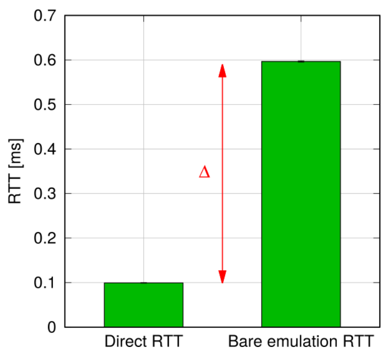

FIGURE 13. Emulation overhead.

the sender and receiver sides, both coded in C++. These processes are unaware of the presence of Simu5G, since the host OS takes care of forwarding their packets through Virtual Ethernet (*veth*) interfaces, as depicted in FIGURE 8: when the sender transmits data via a UDP socket by specifying the IP address of veth2 and the port number the receiver is listening to, while the routing table of the host OS is configured to reroute packets destined to veth2 through veth1 and vice versa, the OS forwards them to the Virtual Ethernet interfaces depicted in FIGURE 8. Ethernet frames sent over such interfaces will appear within the emulated network. More complex configurations can also be envisaged like, e.g., having sender and receiver applications located on different hosts, connected to the host running Simu5G through a local network or the public Internet. An exemplary distributed testbed is shown in FIGURE 9, where the network traffic between two applications running on, respectively, hosts B and C flows through a Simu5G instance running on Host A.

The sender application transmits periodic request messages to the receiver application via a UDP socket. 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 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.

The generic emulated network scenario is depicted in FIGURE 10 and includes one gNB and one FG UE, called the *endpoint UE* henceforth. The latter is the counterpart of the sender process in the real host, as it is endowed with an *ExtInterface* module to capture packets coming from the application. Likewise, the router has one *ExtInterface* module that captures packets coming from the receiver process. With this configuration, packets sent by the sender appear into the emulation at the UE and are forwarded towards the router, which in turn sends them outside the emulation. Then, the OS takes care of forwarding them towards the receiver process. The reverse path is traversed by response messages sent by the receiver process and directed to the sender. Moreover, a number of *simulated* UEs and gNBs can be added to the network to create a more realistic scenario: these UEs communicate with the simulated server, generating traffic that remains *within* the simulator. Unless explicitly stated otherwise, the endpoint UE is 50m far from the gNB. As far as the channel model is concerned, path loss is computed according to the Urban Macro scenario in [15]. Fading and shadowing effects are also considered: the former follows the Jakes model, whereas the latter is modelled according to a log-normal distribution with standard deviation $\sigma_{LOS} = 4$dB and $\sigma_{NLOS} = 6$dB for UEs in line-of-sight and non-line-of-sight, respectively. The main parameters of the network are the ones shown in Table 1.

In order to run the testbed, the receiver process could be launched at any time, since it will remain idle, listening to incoming connections. On the other hand, the sender process needs to start the traffic only after both the receiver and the network emulation has been started. In order to automatize the process, we created a simulation automation tool called

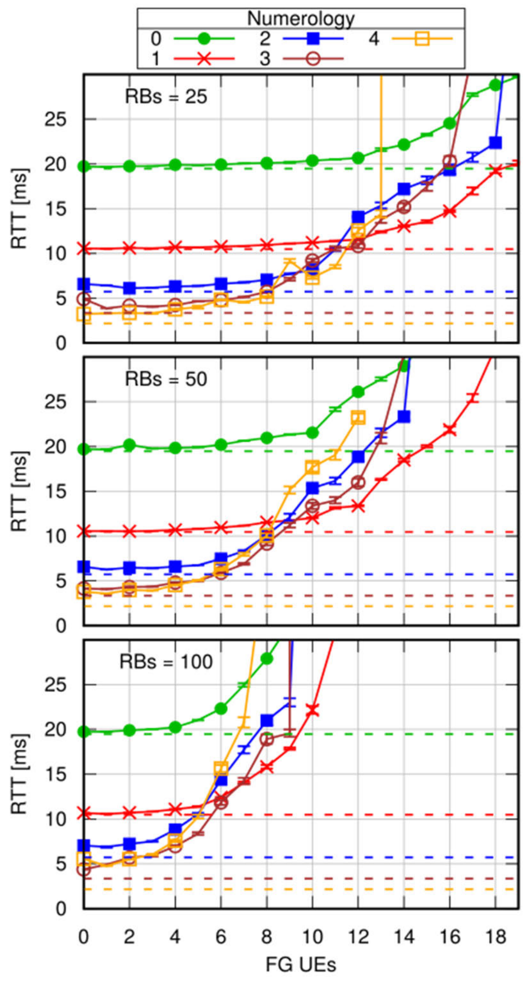

FIGURE 14. RTT analysis, one component carrier with varying number of RBs.

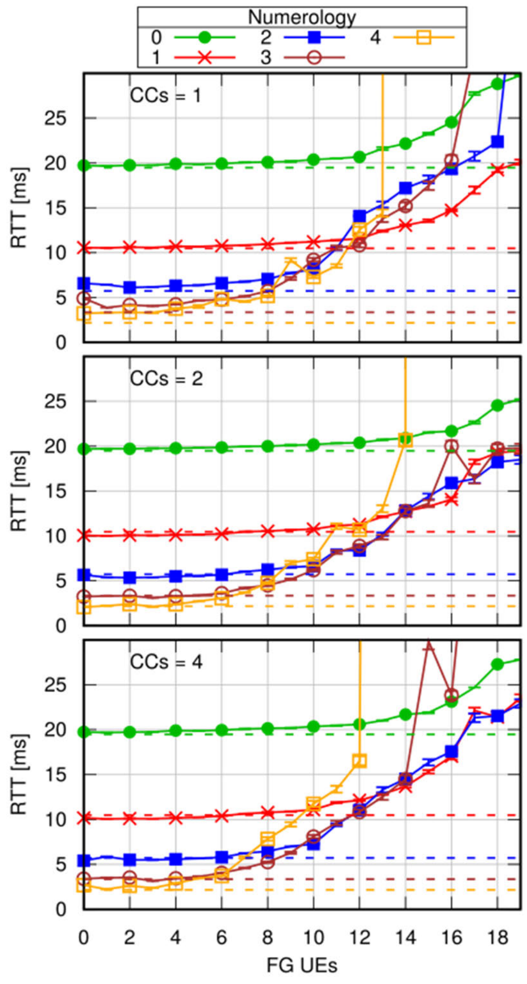

FIGURE 15. RTT analysis, 25 RBs per carrier, varying the number of CCs.

launcher, coded in Perl, that performs the operations depicted in FIGURE 11. It accepts parameters such as the number of simulated UEs and gNBs, 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 creates two new processes for the receiver and sender, respectively. The former starts waiting for request messages from the sender, whereas the latter starts the real data traffic.

B. EXPERIMENTAL RESULTS

In all the following experiments, we make sure that the number of packets sent and received is the same, and we measure the real-time RTT to understand when the emulation starts struggling, depending on the scenario.

In order to be accurate, we first need to quantify the overhead introduced by physical/OS delays in the testbed, as well as the delay due to OS/OMNeT++ interfacing (i.e., the time it takes to capture packets from the real host's OS and inject them into the emulation). We do this by measuring the real-time RTT in the following setup: we run a very simple OMNeT++ emulation, consisting of an INET router that forwards packets between its two infinite-speed interfaces. On one hand, the emulated network is very simple, hence we can expect any real-time delay due to its processing overhead on the host machine to be negligible. On the other hand, packets are not delayed *within* the emulated network (because of infinite-speed interfaces). Thus, the real-time RTT in this scenario will be due to the effects that we want to measure. The setup is depicted in FIGURE 12. As a comparison, we also measure the real-time RTT on the *direct path*, where the sender and receiver applications communicate via the loopback interface of the host. We compute the average of 300 requests, considering a sending interval $T = 1s$. Since the measured real-time RTT is considerably smaller than T, we can assume that the obtained values are independent. We also set $L_{req} = L_{resp} = 1000B$. The results are shown in FIGURE 13. The overhead introduced by traversing an emulated OMNeT++ network is around $\Delta = 0.5$ ms. The 95% confidence intervals, barely visible in the figures,

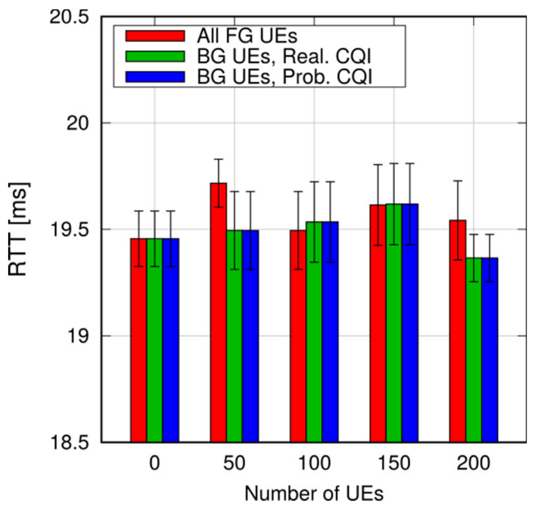

FIGURE 16. Validation of BG UEs, RTT comparison.

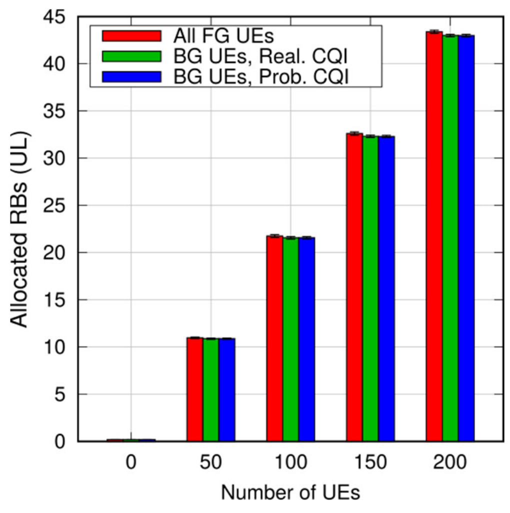

FIGURE 18. Validation of BG UEs, allocated RBs in the uplink.

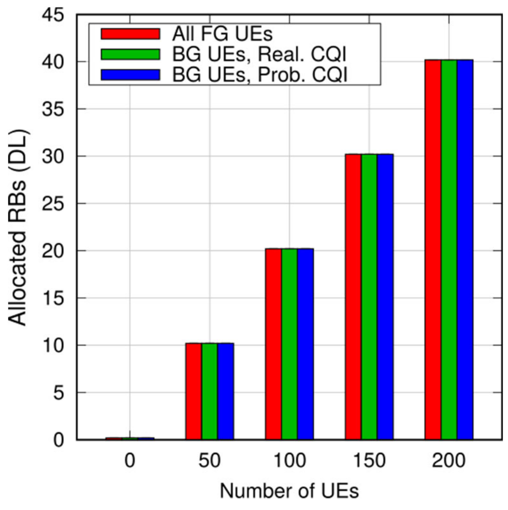

FIGURE 17. Validation of BG UEs, allocated RBs in the downlink.

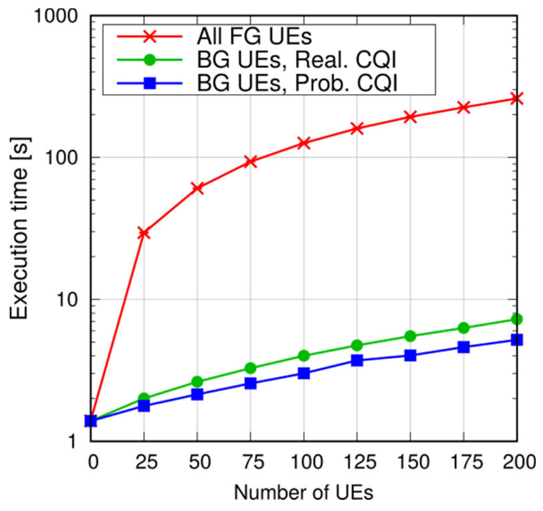

FIGURE 19. Comparison of execution times for the scenarios.

are negligible, which testifies that this overhead is almost constant.

We now assess the scalability of emulation. We run an emulated 5G network with one gNB and an increasing number of simulated UEs (besides the endpoint UE). Initially, all UEs are modeled as FG UEs. The external applications are the same as for the previous experiment.

We measure the real-time RTT (call it R) of packets sent by the sender application as they travel to the receiver and back, traversing the emulated network twice. We need a baseline to compare R to. To this end, we run the *same* scenario within Simu5G in *simulated* mode (i.e., without real-time scheduling of events), with the *same* application code, and measure the simulated-time RTT of packets within Simu5G, call it S. Now, S is a *pessimistic* baseline for R (i.e., $R > S$), because S does not include the overhead Δ measured above. However, on one hand, using $S + \Delta$ as a baseline would instead be optimistic, since there is no evidence that the two times are additive (there are many performance-enhancing mechanisms in a hardware/software system that may leverage parallelism to make the total time less than the sum of the two). On the other hand, we will soon show that $S \gg \Delta$, at least while $\mu \leq 2$. Therefore, we will stick to S as a baseline henceforth. In the above scenario, we vary the number of simulated FG UEs N. Each of them generates the same request-response traffic as the real application, i.e., one 1000-byte packet every 40ms, resulting in a 200kbps traffic per UE. Periods of different applications are desynchronized in the time domain by randomly selecting the starting time of the application. Strictly speaking, S is a random variable. However, the endpoint UE is close to the antenna, and the cell is underloaded in all the simulation settings ($N < 20$). For this reason, S has a negligible variance, and we plot its mean value henceforth.

FIGURE 14 shows R with an increasing number of RBs M and simulated FG UEs N (besides the endpoint UE).

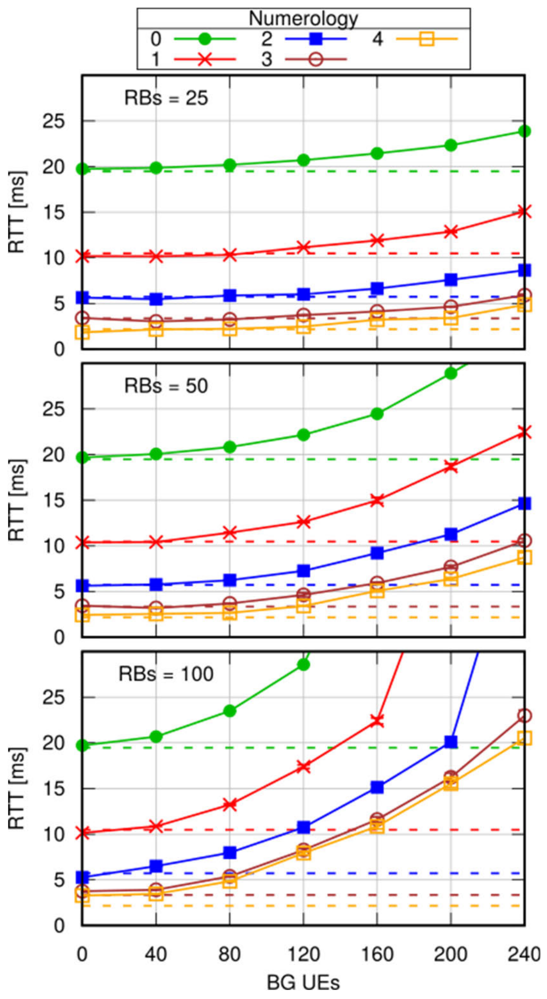

FIGURE 20. RTT analysis, deployment with BG UEs, realistic CQI.

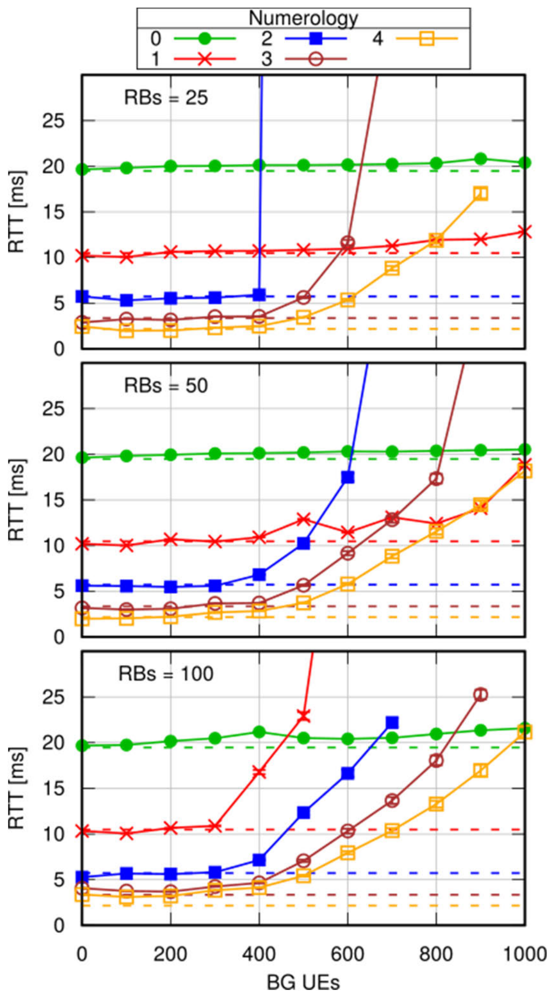

FIGURE 21. RTT analysis, deployment with BG UEs, probabilistic CQI.

Values are reported for all the admissible numerologies. The corresponding S is plotted as a dashed line. In these experiments, carrier aggregation is not employed (i.e., a single carrier is emulated). The figure shows that when M is small, e.g., 25, and $\mu = 0$ or $\mu = 1$ are used, R is basically constant while $N \leq 10$, whereas such threshold is lower when higher numerologies are used. For a larger M, coherence cannot be maintained even for low values of N. Again, the threshold depends on the numerology used. This can expected, since higher numerologies means shorter TTIs (i.e., 1ms long for $\mu = 0$ and 62.5μs long for $\mu = 4$), which makes the emulation more challenging – Simu5G handles in fact 16× events per unit of time when going from $\mu = 0$ to $\mu = 4$ – and coherence is only preserved for small values of M and N.

FIGURE 15 reports the results of the same analysis in a scenario where the same number of RBs is split among different CCs. Each CC has 25 RBs available, and we simulate carrier aggregation with 1, 2, and 4 CCs, hence each graph in FIGURE 15 relates to a scenario with the same overall RBs as the corresponding one in FIGURE 14. In order to balance the load among CCs, we allow one simulated UE to use only one CC, and each CC is assigned the same percentage of simulated UEs. Although the charts confirm that the real-time RTT depends on M and N as in FIGURE 14, we note that coherence is generally preserved for a larger N and a higher μ. In fact, we are now able to perform a real-time emulation with a few simulated UEs also using 100 RBs split into four CCs and $\mu = 4$. This is because each CC has its own independent MAC-level scheduler, according to the carrier aggregation model implemented in Simu5G. On one hand, the MAC layer of the gNB runs more scheduling processes on each TTI, i.e., one per CC. On the other hand, scheduling has super-linear complexity with N. Thus, running more schedulers with smaller M and N makes per-TTI operations is faster than having one scheduler managing all the available RBs and UEs, as in FIGURE 14.

Some comments are in order. The first one is that the above results prove that it is actually possible to run a 5G emulation on a desktop machine, in an environment with a gNB and several FG UEs, whose entire network stack is modeled, from the application to the physical layer. Our second comment is that it is fairly evident that the combinations of number of UEs, RBs, CCs and numerologies with which real-time coherence is preserved are few, and this is quite constraining.

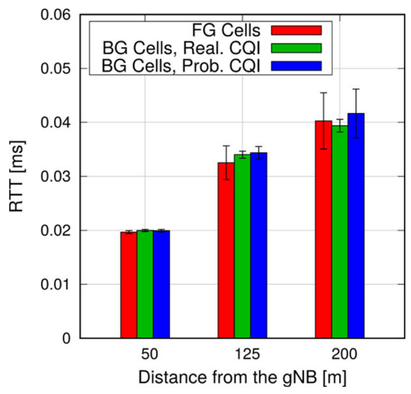

FIGURE 22. Validation of BG cells, RTT comparison with increasing distance from the gNB.

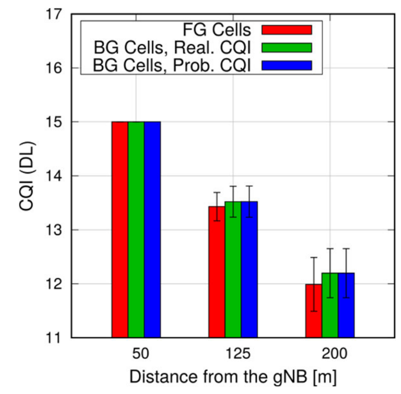

FIGURE 23. Validation of BG cells, DL CQI for the endpoint UE with increasing distance from the gNB.

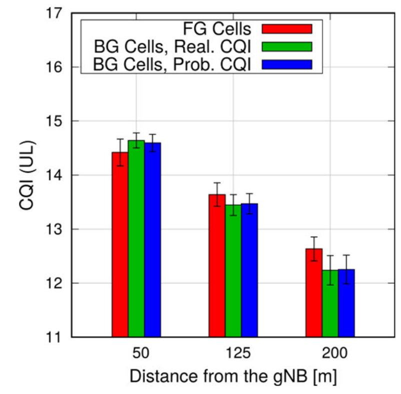

FIGURE 24. Validation of BG cells, UL CQI for the endpoint UE with increasing distance from the gNB.

On one hand, we observe that occasional, slight lapses of coherence (e.g., a moderate, bounded divergence of R from S – this happens when the emulation experiences occasional, temporary losses of coherence and a few packet transmissions suffer additional latency) do not make the emulation worthless: timing properties of the emulated traffic are still fairly reliable even if they may not always be accurate at the TTI level. On the other, this justifies the need for modeling of BG UEs and cells as described in Section III. In the next

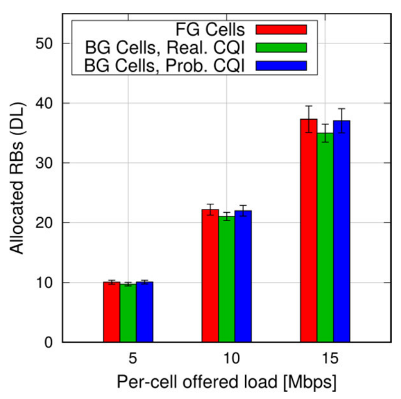

FIGURE 25. Validation of BG cells, allocated RBs in the DL with increasing offered load.

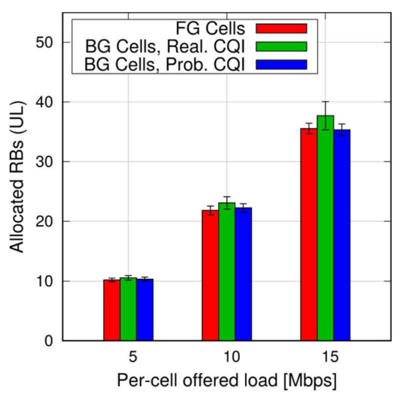

FIGURE 26. Validation of BG cells, allocated RBs in the UL with increasing offered load.

subsection we describe what improvement these bring, in terms of scalability.

C. IMPROVING SCALABILITY WITH BACKGROUND UEs

We now show that employing BG UEs warrants a huge increase in scalability, without any loss in accuracy for the emulated application on the endpoint UE. To demonstrate that the results obtained by an emulation with BG UEs are credible, we simulate a scenario with one gNB serving N UEs, plus the endpoint UE. We assume one CC with $\mu = 0$. In the above scenario, we perform ten independent replicas of a 20-second simulation and compare both application- and network-level metrics obtained when the N UEs are either FG or BG. When using BG UEs, we simulate both realistic and probabilistic CQI computation. For a fair comparison when using the latter method, the mean and variance for each BG UE's CQI is assigned equal to the mean and variance extracted from the simulation with FG UEs. All the metrics are shown for an increasing N and with 95% confidence intervals.

FIGURE 16 shows the simulated-time RTT perceived by the application residing on the endpoint UE. Despite the increasing contention at the scheduler, the simulated-time RTT confidence intervals are overlapping, i.e., the slight variations are not statistically significant. FIGURE 17 and FIGURE 18 show that the scheduler allocates the same number of RBs on average, in both the DL and UL directions, regardless of whether UEs are FG or BG. This demonstrates the accuracy of the modeling of BG UEs and their interaction with the scheduler of the gNB.

FIGURE 19 shows the improvements in terms of execution time of one simulation when replacing FG UEs with BG UEs. We observe that the execution time of the deployment with BG UEs is always smaller than the duration of the simulation, i.e., 20s, also for large values of N. This means that emulating such configurations is feasible, since simulated time runs faster than real time, which is the necessary condition to allow real-time emulations. Instead, when employing the FG for the N UEs, the execution time exceeds the simulation duration when $N \geq 25$, coherently with the results in the previous subsection.

The scalability benefits of using BG UEs are made evident in FIGURE 20, which shows the real-time RTT obtained with the emulation testbed described previously with increasing values of M, N and different numerologies. The graphs refer to the scenario with realistic CQI computation and show that we are now able to emulate tens of UEs. In this scenario, the endpoint UE is close to the gNB (i.e., 50m), hence it always has a high CQI. With MaxC/I scheduling, it is very unlikely that many BG UEs are prioritized over the endpoint UE and add significant scheduling latency. Therefore S (not shown) is constant for all the values of N, M, and any increase in the real-time RTT is due to the overhead of emulation. FIGURE 21 shows the real-time RTT with probabilistic CQI computation. In this case we observe that the real-time RTT is constant with up to 300 UEs for all the configurations. With some of them, e.g., with $\mu = 0$, we can emulate up to 1000 BG UEs. This shows that one can perform real-time emulation of a 5G NR cell serving *hundreds of* UEs.

D. IMPROVING SCALABILITY WITH BACKGROUND CELLS

In this section we show that the proposed model of BG cells provides another degree of freedom to increase the scale of emulation, in addition to using BG UEs. Emulating multi-cell networks makes scenarios more realistic by bringing the effects of inter-cell interference into the picture.

We demonstrate that BG cells do not impair the accuracy of the emulation. We consider a scenario where one gNB serves the endpoint UE and N BG UEs. Moreover, we simulate $B = 6$ gNBs surrounding the above gNB, each serving N BG UEs. We assume that the latter generate 200kbps traffic each, in both the UL and DL directions, and the gNBs have one CC with $\mu = 0$ and $M = 100$ RBs. While the central gNB is mandatorily a FG cell (because packets sent and received by the endpoint UE must traverse the whole NR protocol stack and be affected accordingly), the surrounding B gNBs can be either FG or BG cells. Thus, we compare the results obtained in those two scenarios, by simulating ten independent replicas of a 20-second simulation and showing average metrics with 95% confidence interval. Like we did in previous section,

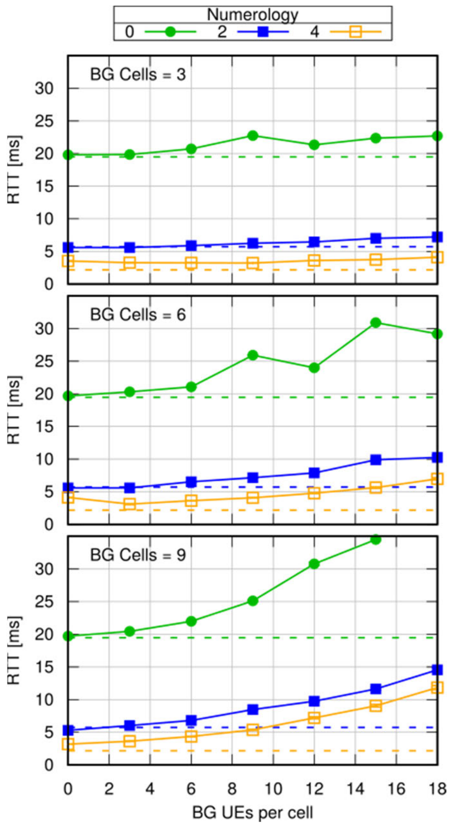

FIGURE 27. Real-time RTT analysis, deployment with BG cells, realistic CQI.

we consider both realistic and probabilistic CQI computation for BG UEs. We consider several network configurations (i.e., by varying the distance between the endpoint UE and the gNB, as well as cells' offered load) and show that the results in terms of different metrics are the same when using either FG or BG cells.

We first fix $N = 50$, resulting in an offered load of 10Mbps per cell, and consider increasing distance between the endpoint UE and its serving gNB, i.e., 50, 125 and 200m. FIGURE 22 shows the simulated-time RTT perceived by the application on the endpoint UE. The chart shows that the simulated-time RTT increases with the distance, as the channel quality between the UE and its serving gNB degrades. This can be observed in FIGURE 23 and FIGURE 24, which show the CQI for the endpoint UE in the DL and UL directions, respectively. In all the three graphs, the variations among the three scenarios are not statistically significant, which validates our models with BG cells. To validate the number of allocated RBs, we now change the offered load for all the cells, ranging from 5 to 15 Mbps, assuming that the endpoint UE is 125m far from its gNB. FIGURE 25 and FIGURE 26 report the allocated RBs by the central gNB in DL and UL, respectively. Also in this case, we can conclude that there are no significant differences in using FG and BG cell, hence modeling cells as BG cells preserves the accuracy of the emulation.

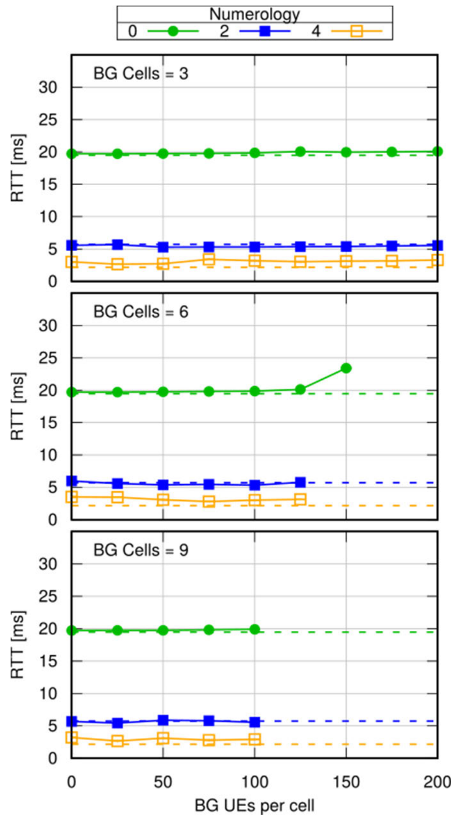

FIGURE 28. Real-time-RTT analysis, deployment with BG cells, probabilistic CQI.

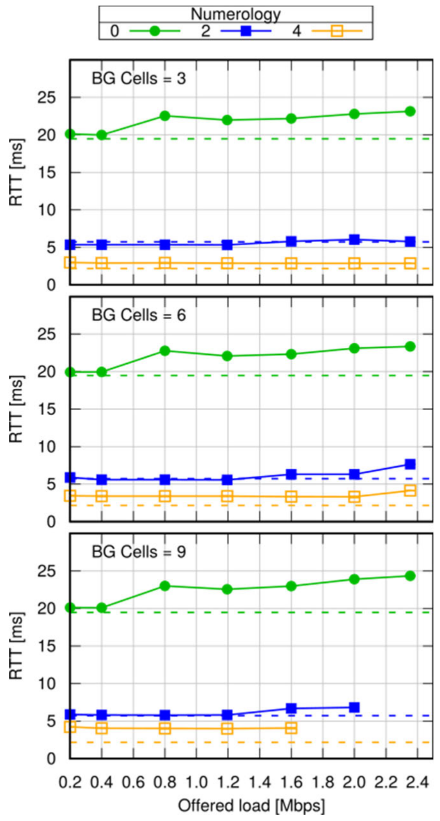

FIGURE 29. Real-time RTT analysis, deployment with BG cells and increasing endpoint UE's offered load.

We now assess the benefits of BG cells in terms of emulation scalability using the emulation testbed. FIGURE 27 and FIGURE 28 show the real-time RTT with increasing B and different values of μ. FIGURE 27 plots results when BG UEs employ realistic CQI computation, whereas FIGURE 28 reports results with probabilistic CQIs. The endpoint UE is now 50m far from the gNB, hence variations in the real-time RTT can be ascribed to distortions introduced by the real-time emulation only. When realistic CQI computation is used, we are able to emulate up to 20 BG UEs per cell with minimal deviations when $B = 3$. For a larger B the emulation becomes challenging, and this confirms that PHY-layer processing is a scalability bottleneck. Instead, when probabilistic CQI is employed, we avoid most of the time-consuming operations for computing SINR of BG UEs and the emulation can scale up to tens of BG UEs per cell. For example, with $\mu = 4$ we can emulate three BG cells with 200 BG UEs each, or nine BG cells with 100 BG UEs each.

Finally, we evaluate the real-time RTT when varying the offered load of the endpoint UE, i.e., the real traffic injected by the external application into the emulation. We fix $M = 100$ and $N = 100$, which has been shown to be one of the most challenging configurations (see FIGURE 28). The overall traffic load generated by background UEs in each cell is 20 Mbps. FIGURE 29 shows the real-time RTT when varying the application-level sending rate of the endpoint UE. Results are shown for three, six and nine BG cells, and with $\mu \in \{0, 2, 4\}$. In this scenario, preserving the accuracy is more challenging when $\mu = 0$. This is not surprising, since this is the configuration with the highest frequency resource utilization, hence the scheduler has to perform more operations. However, we note that variations are in the order of few milliseconds and this might be still tolerable if the objective of the emulation is to evaluate, e.g., other QoS/QoE metrics for the application instead of its RTT with high level of detail. With larger μ, the real-time RTT stays constant when three BG cells are employed, whereas it starts to increase at higher offered loads when the number of BG cells is six or nine. In the most challenging configuration, i.e., nine BG cells and $\mu = 4$, we can emulate scenarios where the offered load generated by the endpoint UE is up to 1.6 Mbps.

V. RELATED WORKS

To the best of our knowledge, there are few works that can compare to the one described in this paper. As we argue in [3], *physical-layer simulators,* such as the 5G K-Simulator [18] suite or the Vienna 5G SL simulator [19], are designed with different purposes, namely evaluating physical-layer quantities, such as the Signal to Interference-plus-Noise Ratio (SINR) or spectral efficiency, as a function of physical-layer designs, such as antenna layout, transmission schemes. They are often very accurate in the modeling of the physical layer, but they seldom, if ever, allow end-to-end application-level communication through the whole protocol stack. On the other hand, *end-to-end* simulators, such as Simu5G, are designed exactly for the last purpose. As a consequence, they often model the physical layer in such a way that the *effects* of physical impairments on packet transmission are preserved, without necessarily modeling symbol transmission, constellations, receiver algorithms, etc. Among end-to-end simulators of 5G networks, we find 5G-LENA [20] and the 5G-air-simulator [21]. The former includes a subset of Simu5G's features (e.g., it lacks FDD, dual connectivity, D2D, MEC modeling, etc.). The second one lacks some of the above too, and does not incorporate models of layer-4 protocols, which makes it impossible (e.g.) to simulate anything TCP-based with the necessary credibility. To the best of our knowledge, there is no mention of either of the above working in real time. We are aware that ns3, on which 5G-LENA is based, allows real application to interact with simulated models, and can also run in real time (see, e.g., [28], [29], which discuss LTE emulation). However, this does not imply that 5G-LENA is lightweight enough to run in real time, nor it sheds any light on the scale – if any – at which this would be possible. We are not aware of any work assessing feasibility and scalability of real-time emulation of 5G networks using 5G-LENA.

A previous work of ours [1] introduced the theme of real-time emulation with Simu5G. However, that work relies on outdated versions of Simu5G, INET and OMNeT++. Simu5G's version in [1] relied on version 3.6.4 of the INET library, which did not exhibit the same performance as version 4.3, on which this work is based. The differences are many, and in fact involve the mechanism for interfacing with external applications. All the above translated to improved baseline performance: for instance, it is now possible – and it was not in [1] – to run emulation with FG UEs at high numerologies. Moreover, in this work we have added modeling of BG UEs/gNBs, which allows Simu5G emulation to scale up considerably.

A related effort is the OpenAirInterface (OAI) 5G-RAN project [26], which is currently under way as we write, with a schedule foreseen to complete in the second half of 2022. The project promises to deliver an open-source software implementation of a programmable 5G RAN. With respect to that project, Simu5G maintains the distinct advantage of allowing both simulation and emulation with the same codebase. This allows one, for instance, to modify elements in a whole (simulated/emulated) network scenario with few modifications of an INI file, rather than (e.g.) by recompiling code on several physical/virtual machines. Moreover, Simu5G allows full control over the simulated scenario, which is constructed by the user, whereas it is unlikely that OAI 5G-RAN will.

Two simulation software packages exist whose features are at least comparable to Simu5G's, namely Riverbed Modeler [16] and Tetcos Netsim [17]. As far as we are aware, Riverbed Modeler does not include any 5G NR module (the website mentions LTE release 8 and 9), and there is no indication that it allows real-time emulation. Tetcos Netsim instead allows emulation, and it has 5G mm-wave NewRadio access. However, both are only included in the standard, non-academic version, which is not free. Finally, commercial emulation software packages exist, aimed at network testing, e.g., Simnovus Callbox [22], VIAVI TeraVM 5G Core Test [23], NetTest 5G Network Emulators [24], Keysight Technologies 5G products [25]. Their focus is mainly on equipment testing, hence is different from ours.

VI. CONCLUSION AND FUTURE WORK

In this paper we have assessed the real-time emulation capabilities of Simu5G, a novel 5G New Radio simulator based on OMNeT++ and INET. We introduced *lightweight* models of UEs and gNBs, which allow a user to run a scenario where UL/DL resource contention and interference are correctly represented, without paying the price of a fully-fledged multicell simulation: we showed that using lightweight models greatly increases the scale at which emulation is possible without sacrificing the accuracy in the least. We have shown that it is possible to emulate in real time a multicell 5G network with ten cells and a thousand UEs, carrying application traffic exceeding 1.6Mbps, over an off-the-shelf desktop PC.

The work presented in this paper describes a new use for Simu5G: beside the classical one as a tool for researchers interested in evaluating network performance (e.g., comparing scheduling algorithms), we now add that of *5G testbed* for application developers to test and demonstrate their applications, using inexpensive hardware. With respect to this, there is more work being developed at the time of writing: we are finalizing the addition of ETSI-compliant MEC models to Simu5G, complete with standardized interfaces towards the application endpoints. With Simu5G running the MEC infrastructure in real time, a MEC app developer will be able to test a MEC app over a mixed communication/computation emulated infrastructure, in controllable experimental conditions (e.g., network load, channel quality, resource contention or availability of MEC services at the MEC host etc.), using the same codebase used for production.

At the time of writing, Simu5G is also being used in the framework of the Hexa-X EU project [8]. In that framework, it will support the development, testing and demonstration of federated learning of Explainable Artificial Intelligence (XAI) models. In that context, the required network functions and protocols will be added to Simu5G, and will interface with external applications performing the learning task.

ACKNOWLEDGMENT

An earlier version of this paper was presented in part at the 2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications [DOI: 10.1109/PIMRC48278.2020.9217177].

REFERENCES

[1] G. Nardini, G. Stea, A. Virdis, D. Sabella, and P. Thakkar, "Using Simu5G as a realtime network emulator to test MEC apps in an end-to-end 5G testbed," in *Proc. PiMRC*, London, U.K., Sep. 2020, pp. 1–7.

[2] A. Virdis, G. Stea, and G. Nardini, "Simulating LTE/LTE-advanced networks with SimuLTE," in *Advances in Intelligent Systems and Computing*, vol. 402. Cham, Switzerland: Springer, Jan. 2016, pp. 83–105, doi: 10.1007/978-3-319-26470-7_5.
[3] 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. 181176–181191, 2020, doi: 10.1109/ACCESS.2020.3028550.
[4] *Simu5g Website*. Accessed: Jun. 2021. [Online]. Available: http://simu5g.org
[5] *OMNeT++ Website*. Accessed: Jun. 2021. [Online]. Available: https://omnetpp.org
[6] *INET Library Website*. Accessed: Jun. 2021. [Online]. Available: https://inet.omnetpp.org
[7] *VEINS Simulator Website*. Accessed: Jun. 2021. [Online]. Available: https://veins.car2x.org/documentation/modules/
[8] *Hexa-X project Website*. Accessed: Jun. 2021. [Online]. Available: http://hexa-x.eu/
[9] *NR; Multi-Connectivity; Overall Description; Stage-2 (Release 16)*, Standard 3GPP TS 37.340 v16.5.0, Mar. 2021.
[10] G. Nardini, A. Virdis, and G. Stea, "Modeling network-controlled device-to-device communications in SimuLTE," *Sensors*, vol. 18, no. 10, p. 3551, Oct. 2018, doi: 10.3390/s18103551.
[11] C. Yahiaoui, M. Bouhali, and C. Gontrand, "Simulating the long term evolution physical layer," in *Proc. 17th EUSIPCO*, Glasgow, U.K., Aug. 2009, pp. 1471–1478.
[12] C. Sommer, R. German, and F. Dressler, "Bidirectionally coupled network and road traffic simulation for improved IVC analysis," *IEEE Trans. Mobile Comput.*, vol. 10, no. 1, pp. 3–15, Jan. 2011.
[13] A. Hegde and A. Festag, "Artery-C: An OMNeT++ based discrete event simulation framework for cellular V2X," in *Proc. 23rd Int. ACM Conf. Modeling, Anal. Simulation Wireless Mobile Syst.*, Nov. 2020, pp. 47–51, doi: 10.1145/3416010.3423240.
[14] *Study on New Radio Access Technology Physical Layer Aspets*, Standard 3GPP TR 38.802 v14.2.0, Sep. 2017.
[15] *Study on 3D Channel Model for LTE*, Standard 3GPP TR 36.873 v12.7.0, Dec. 2017.
[16] *Riverbed Modeler Website*. Accessed: Jun. 2021. [Online]. Available: https://www.riverbed.com/gb/products/npm/riverbed-modeler.html
[17] *Tetcos Netsim Website*. Accessed: Jun. 2021. [Online]. Available: https://www.tetcos.com/netsim-acad.html
[18] Y. Kim, J. Bae, J. Lim, E. Park, J. Baek, S. I. Han, C. Chu, and Y. Han, "5G K-simulator: 5G system simulator for performance evaluation," in *Proc. IEEE Int. Symp. Dyn. Spectr. Access Netw. (DySPAN)*, Oct. 2018, pp. 1–2, doi: 10.1109/DySPAN.2018.8610404.
[19] 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 J. Wireless Commun. Netw.*, vol. 2018, no. 1, p. 227, Dec. 2018, doi: 10.1186/s13638-018-1238-7.
[20] N. Patriciello, S. Lagen, B. Bojovic, and L. Giupponi, "An E2E simulator for 5G NR networks," *Simul. Model. Pract. Theory*, vol. 96, Nov. 2019, Art. no. 101933, doi: 10.1016/j.simpat.2019.101933.
[21] S. Martiradonna, A. Grassi, G. Piro, and G. Boggia, "5G-air-simulator: An open-source tool modeling the 5G air interface," *Comput. Netw.*, vol. 173, May 2020, Art. no. 107151.
[22] *Simnovus Callbox Website*. Accessed: Jun. 2021. [Online]. Available: https://simnovus.com/products/callbox/
[23] *Viavi Solutions Website*. Accessed: Jun. 2021. [Online]. Available: https://www.viavisolutions.com/
[24] *Polaris Networks 5G Emulators Website*. Accessed: Apr. 2021. [Online]. Available: https://www.polarisnetworks.net/5g-network-emulators.html
[25] *Keysight Technologies Website*. Accessed: Jun. 2021. [Online]. Available: https://www.keysight.com/
[26] *OpenAirInterface 5G Ran project Website*. Accessed: Jun. 2021. [Online]. Available: https://openairinterface.org/oai-5g-ran-project/
[27] S. Tavakoli, A. Mousavi, and A. Komashie, "A generic framework for real-time discrete event simulation (DES) modelling," in *Proc. Winter Simulation Conf.*, Dec. 2008, pp. 1931–1938, doi: 10.1109/WSC.2008.4736285.
[28] R. Gupta, B. Bachmann, R. Ford, S. Rangan, N. Kundargi, A. Ekbal, K. Rathi, M. I. Sanchez, A. de la Oliva, and A. Morelli, "Ns-3-based real-time emulation of LTE testbed using LabVIEW platform for software defined networking (SDN) in CROWD project," in *Proc. Workshop Ns-3*, May 2015, pp. 91–97, doi: 10.1145/2756509.2756516.
[29] T. Molloy, Z. Yuan, and G.-M. Muntean, "Real time emulation of an LTE network using NS-3," in *Proc. 25th IET Irish Signals Syst. Conf. China-Ireland Int. Conf. Inf. Communities Technol. (ISSC /CIICT)*, 2014, pp. 251–257, doi: 10.1049/cp.2014.0694.
[30] A. Virdis, G. Nardini, G. Stea, and D. Sabella, "End-to-end performance evaluation of MEC deployments in 5G scenarios," *J. Sensor Actuator Netw.*, vol. 9, no. 4, p. 57, Dec. 2020, doi: 10.3390/jsan9040057.
[31] M. Carson and D. Santay, "NIST Net: A Linux-based network emulation tool," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 111–126, Jul. 2003, doi: 10.1145/956993.957007.
[32] D. Mahrenholz and S. Ivanov, "Real-time network emulation with ns-2," in *Proc. 8th IEEE Int. Symp. Distrib. Simulation Real-Time Appl.*, Oct. 2004, pp. 29–36, doi: 10.1109/DS-RT.2004.34.
[33] B. D. Noble, M. Satyanarayanan, G. T. Nguyen, and R. H. Katz, "Trace-based mobile network emulation," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 1997, pp. 51–61, doi: 10.1145/263105.263140.

GIOVANNI NARDINI received the M.Sc. degree *(summa cum laude)* in computer systems engineering and the Ph.D. degree in information engineering from the University of Pisa, in 2013 and 2017, respectively. He is currently an Assistant Professor with the Department of Information Engineering, University of Pisa. In these fields, he has coauthored more than 30 peer-reviewed articles and holds six patents. His research interests include resource allocation algorithms for quality of service in 4G and 5G networks, multi-access edge computing, and the simulation and performance evaluation of computer networks. He has been involved in EU-funded and industrial research projects.

GIOVANNI STEA received the Ph.D. degree from the University of Pisa, Italy, in 2003. He is currently an Associate Professor with the Department of Information Engineering, University of Pisa. His current research interests include the quality of service and resource allocation in wireline and wireless networks, performance evaluation through simulation and analytical techniques, and traffic engineering. In these fields, he has coauthored more than 100 peer-reviewed articles and holds 16 patents. He has been involved in the national and European research projects, and he has led joint research projects with industrial partners. He is a fellow of the European Alliance for Innovation. He has served as a member of the technical and/or organization committees for several international conferences, including SIGCOMM, WoWMoM, and VALUETOOLS, and he is serving on the editorial board for *Wireless Networks* journal.

ANTONIO VIRDIS received the M.Sc. degree in computer system engineering and the Ph.D. degree in information engineering from the University of Pisa, in 2011 and 2015, respectively. He is currently an Assistant Professor with the University of Pisa. He has edited the book titled "*Recent Advances in Network Simulation*," published by Springer/EAI. His research interests include the quality of service, edge computing, network simulation, and performance evaluation. He has coauthored more than 60 peer-reviewed articles and holds eight patents in the above fields. He is/has been involved in research projects supported by private industries and funded by the EU community. He served as the TPC Chair for the International OMNeT++ Summit and for the IEEE SmartSys workshop. He is one of the authors and maintainers of the SimuLTE and Simu5G opensource projects, for the system-level simulation of 4G and 5G communication networks.